SAMPLING BASED REAL-TIME APPROXIMATIONS ON CPUS AND GPUS

with application to rendering of multi layered materials and path finding

Von der Universität Bayreuth
zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

von
MAXIMILIAN MICHAEL DOMINIKUS REISCHL
aus Wasserburg am Inn

1. Gutachter: Prof. Dr. Michael Guthe
2. Gutachter: Prof. Dr. Jiří Bittner

Tag der Einreichung: 26.10.2023
Tag des Kolloquiums: 27.02.2024

Maximilian Michael Dominikus Reischl: *Sampling based real-time approximations on CPUs and GPUs,* with application to rendering of multi layered materials and path finding

The only way to deep happiness is
to do something you love
to the best of your ability.

— Richard Feynman

Ohana means family.
Family means nobody gets left behind or forgotten.

— Lilo & Stitch

Dedicated to Rike, Erik, Arthur and Robin.
You have been my unwavering source of love, encouragement and strength. Through every challenge and triumph, your support has been my foundation. This achievement is as much yours as it is mine, and I dedicate this work to those who inspire me every day.

Max

ABSTRACT

Estimation and approximation are essential parts of the history of electronic computing, especially in simulation and games. In this work, we present two new approaches to problems that are not solvable in real-time by the methods normally used, and show how our approaches achieve this in a scalable way with minimal constraints, as well as incorporating GPUs for computation. First, we establish a method that enables real-time volume scattering in multilayer materials for entertainment and medical applications. Second, one that enables the computation of paths for a variety of agents with minimal precomputation for entertainment technology and simulations, also in real-time.

ZUSAMMENFASSUNG

Näherung und Schätzungen sind wesentliche Bestandteile der Geschichte der elektronischen Datenverarbeitung, insbesondere im Bereich der Simulation und der Spiele. In dieser Arbeit stellen wir zwei neue Ansätze für Probleme vor, die mit den normalerweise verwendeten Methoden nicht in Echtzeit lösbar sind, und zeigen, wie unsere Ansätze dies auf skalierbare Weise mit minimalen Einschränkungen erreicht sowie GPUs zur Berechnung einbezieht. Zum einen etablieren wir ein Verfahren, das Volumenstreuung in mehrlagigen Materialien für Unterhaltungstechnik und medizinische Anwendungen in Echtzeit ermöglicht. Zum anderen eines, das die Berechnung von Pfaden für eine Vielzahl von Agenten mit minimaler Vorberechnung für Unterhaltungstechnik und Simulationen ebenfalls in Echtzeit ermöglicht.

CONTENTS

1 INTRODUCTION

Many successes in computer history can be attributed to approximations and estimates. The very basis of many calculations, the floating point numbers, are inherently imprecise. The finite element method, to name a well-known method that was developed long before the invention of the computer, made today's means of transportation possible. The field of computer games has greatly benefited from approximations. Inaccuracies were of little consequence on the way to today's photorealistic computer games, and a simplified representation did not detract from the gaming experience. In turn, computer games have contributed to the further development and widespread distribution of computers. Contrary to earlier predictions, households now own dozens of computers.

1.1 TIME CRITICAL COMPONENTS IN COMPUTER GAMES

Nowadays, there are usually three time-critical components in computer games: Graphics, pathfinding and physics, where pathfinding is usually grouped with behavior as AI. All of these components can have a direct impact on the quality of gameplay. Like physics, graphics performance is crucial for a realistic perception of the game world, while pathfinding is crucial for the game characters' ability to navigate through complex game environments. The more complex and detailed the game environments become, the more difficult it is to optimize graphics and pathfinding.

1.1.1 *Approximations in computer graphics*

In addition to the fact that raster images are estimates due to their resolution and meshes are approximations of objects, there are several common techniques that have been essential to the rendering pipeline for decades. One such technique is the Z-buffer, which has led to several improved versions and adaptations, including shadow mapping. Level-of-detail (LOD) rendering and mipmaps are other commonly used optimization techniques in this area. Lighting is a crucial element in graphics as it greatly impacts the perception, quality, and realism of a scene. Various algorithms have been established to handle dynamic lighting, refraction, and scattering, with some dating back decades. These algorithms also consider human perception and adjust the calculations accordingly, as the calculation effort required for these tasks is significant. This is particularly relevant to shadow

calculations and reflections, as well as to newer applications such as the peripheral field of view in VR applications.

1.1.2 *Approximations in pathfinding*

While there are many greedy algorithms, such as the best known, A*, but they only use heuristics to find the best path as quickly as possible and not to estimate it. However, for games and many simulations with large numbers of agents, it is not absolutely necessary to find the optimal paths. The background is that minor detours may not be noticeable or perceived as necessary here, but also that interaction between agents may occur. Below, we will provide further details on these methods.

1.1.3 *Approximations in game physics*

Game physics are not discussed in this section. However, for the sake of completeness, we will briefly touch on them. Rigid bodies are mostly used in games, and there are various numerical integration methods for their simulation, each with its own characteristics and limitations. Realistic simulation of physics is a crucial element in game development, as many game mechanics rely on it, and players have a natural understanding of how objects should move based on their experiences.

1.2 GPGPU

While dedicated hardware for rasterization and partly also for ray tracing already exists and is installed in almost every gaming PC, there is no widespread hardware that supports the CPU in path finding, since these are mainly sequential calculations task of the CPU. Therefore, it is obvious to use the enormous capacities of the GPU for pathfinding, but there are few algorithms that are designed for this. However, with minimal restrictions, methods can be found that don't stand out negatively in the game despite non-optimal paths and are even sufficient for simulations. On the other hand, the current graphics hardware also reaches its limits, especially in the case of scattering and multiple refraction in real time, so that estimates are also applied here which the human eye can hardly perceive and which are very close to reality.

Here we will present two algorithms that are excessively dedicated to estimation and optimization to solve hard problems using the GPU in such a way as to achieve real-world results in real time for both games and simulations.

1.3 RELATED PUBLICATIONS

Parts of the following publications have been integrated into this work:

- Physically Based Real-Time Rendering of Teeth and Partial Restorations [29]
- Parallel Near-Optimal Pathfinding Based on Landmarks [30]
- Using Landmarks for Near-Optimal Pathfinding on the CPU and GPU [31]

2

REALTIME RENDERING OF TEETH

2.1 MOTIVATION

Visually accurate real-time rendering of teeth has many applications ranging from computer games to dental CAD. Similar to skin, the realistic and physically correct appearance of teeth cannot be achieved by simply using opaque diffuse textures, mainly because of the subsurface scattering behaviors of both. While both have a layered structure in common, the scattering characteristics of the teeth layers are drastically different from those of the skin, making rendering much more complicated.

We present an approach which uses the Henyey-Greenstein scattering to achieve a near realistic real-time rendering of human teeth. In order to simulate the multi-layered geometry of teeth, we use standardized teeth models with dentin cores and fit them to real scanned teeth or dental restorations. By using a proxy geometry to compute the scattering, we can also render partial restorations as they would look like when attached to the remaining teeth. Finally, we compare our results to the VITA shade systems and human teeth to evaluate the visual fidelity of our approach.

2.2 RELATED WORK

Subsurface scattering is an important optical effect in most tissues and gaseous phenomena. It has been investigated in different fields of science, e.g. astrophysics [19]. In computer graphics, the diffuse dipole approximation [22] is the most common approach for highly scattering materials. For real-time rendering, this can be combined with a specialized sampling pattern, like the 21-tap kernel of Dachsbacher and Stamminger [6]. Rendering of layered materials [16] is especially important for human tissue like skin or teeth. In addition, the reflection of the saliva – the thin water film on the teeth – has to be considered, e.g. using the approach of Jensen et al. [21]. Highly anisotropic materials exhibit strong directional scattering that can be more faithfully rendered using the directional dipole model [11].

There have been multiple studies for the measurement of the optical properties of teeth, beginning with monochromatic light [38]. More recently, multiple wavelengths have been used in the visible [48, 49] and near infrared spectrum [10]. Fernández-Oliveras et al. measured the scattering anisotropy of enamel and dentin in comparison to other biomaterials [9]. They approximated the measured scattering

profiles of dentin and enamel samples of two different thicknesses using the Henyey-Greenstein phase function [19] yielding a good approximation. These works have shown that scattering in enamel is strongly directional with a reduced scattering coefficient of about 1.3 mm^{-1} at a thickness of up to 3 mm from top and 1 mm from the side of a tooth. Single scattering is however not sufficient at this thickness due to the scattering coefficient of about 4 mm^{-1}. Dentin on the other hand is very strongly scattering with a reduced scattering coefficient of approximately 70 mm^{-1}.

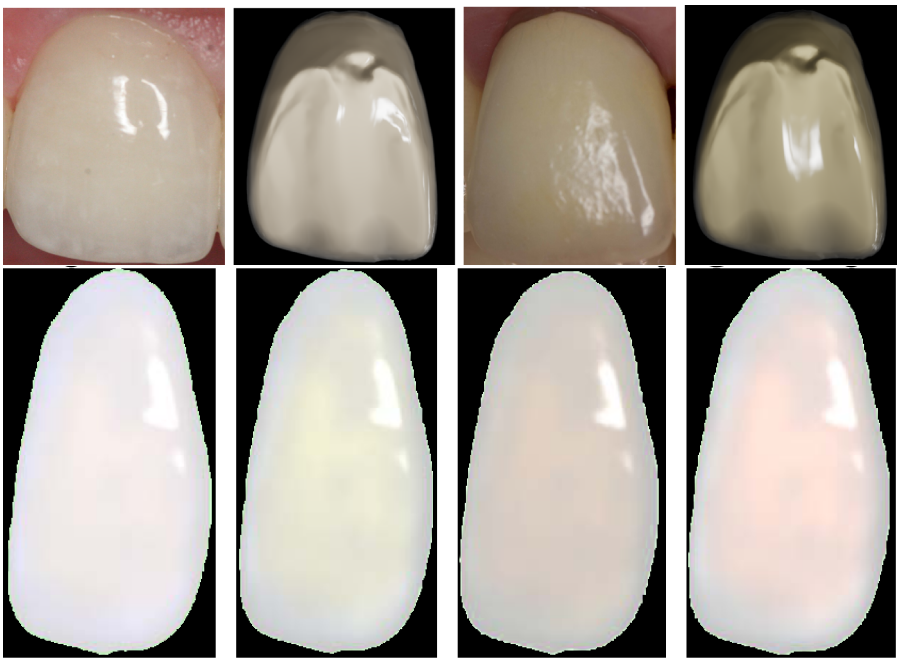


Figure 2.1: The results of Shetty and Bailey (top) and Larsen et al.(bottom).

Some other methods have been directly developed for realistic rendering of human teeth. Shetty and Bailey [36] used the multi-pole diffusion model to render the layered structure for their work. The multi-pole model is however not suitable for such a large range of scattering coefficients. The most recent technique in this area by Larsen et al. [25, 26] also targets at real-time rendering of human teeth for dentists or dental technicians. They used a 12-tap kernel in connection with the diffuse dipole approximation for dentine and single scattering for enamel. The dentin core is generated by shrinking the tooth surface. This approach, however, causes problems for molar teeth and sometimes even for the incisors or canine teeth. Due to the simple space deformation, the enamel thickness is imprecise and can sometimes even become less than zero, i.e. enamel and dentin meshes intersect. For enamel, single scattering requires modification to account for the 5 to 10 scattering events that occur on average, so most of the scattered light is not considered. In addition, the 12-tap kernel is designed for skin and not the highly scattering and thicker

dentine. Finally, they used the wrong anisotropy for enamel, since they did not take the isotropic scattering part into account when using the values from Fried [10].

The dental CAD system into which our algorithm is integrated is based on a deformable model of teeth [1] – also called biogeneric model – that represents a tooth by an "average" model and variations encoded as principal components. For non-medical applications, we could also use more recent methods that generate teeth models from photographs like [46].

2.3 BACKGROUND

Before describing the details of the rendering algorithm, we first review the background of light transport in the teeth.

2.3.1 *Subsurface scattering*

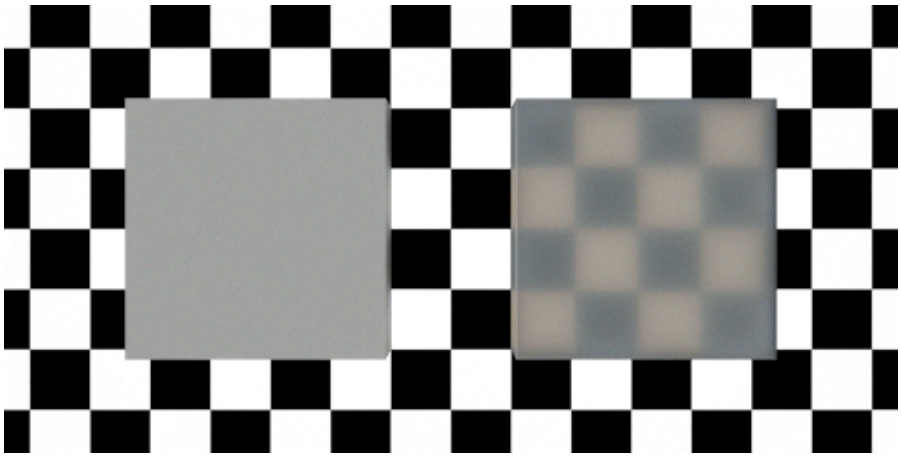


Figure 2.2: Raytraced examples for scattering in dentin ($g = 0.44$) and enamel ($g = 0.68$) with 1mm thickness.

Scattering of light can occur when it travels through a participating medium, where it can be either absorbed or change its direction at each interaction. The angle by which it deviates, is called the scattering angle (θ). While $\theta = 0$ means that the direction stays the same, $\theta = \pi$ results in a total reversed direction. Equation 2.1 shows the Henyey-Greenstein phase function [19], which is widely used for the simulation of scattering in computer graphics and describes the probability $p(\theta)$ of light being scattered in the direction θ. By varying the scattering anisotropy factor g, it ranges from complete backward scattering ($g = -1$) through isotropic scattering ($g = 0$) to complete forward scattering ($g = 1$).

$$p(\theta) = \frac{1}{4\pi} \frac{1 - g^2}{(1 + g^2 - 2g\cos(\theta))^{1.5}} \tag{2.1}$$

The quantity of light being absorbed depends on the distance d the light travels and the absorption coefficient σ_a. The reduced intensity I_o is:

$$I_o = I_i e^{-\sigma_a d} \tag{2.2}$$

When the material also scatters light, the reduced intensity of light that travels through the material without any interaction also depends on the scattering coefficient σ_s:

$$I_o = I_i e^{-(\sigma_a+\sigma_s)d} \tag{2.3}$$

While it is simple to analytically compute absorption, scattering can only be computed by sampling. The most simple, but also most computationally expensive solution is to simulate the path of photons through the material. If the material is homogeneous, the path length l of a photon between two scattering events can be calculated from a uniform random variable φ:

$$l = -\ln \frac{\varphi}{\sigma_s + \sigma_a} \tag{2.4}$$

If an interaction between photon and material occurs, the probability of absorption p_a is:

$$p_a = \frac{\sigma_a}{\sigma_s + \sigma_a} \tag{2.5}$$

Instead of discrete absorption, it is also possible to only simulate scattering and use the analytic absorption (eq. 2.2). The random function for the path length then becomes:

$$l = -\ln \frac{\varphi}{\sigma_s} \tag{2.6}$$

For strongly scattering materials, the diffuse [22] or directional dipole approximation [11] can be used to compute the scattering more efficiently. Instead of simulating the photon paths, an integral over the surface S is calculated:

$$I_o(\mathbf{x_o}, \omega_o) = \int_S \int_\Omega \rho(\mathbf{x_i}, \mathbf{x_i}, \omega_i, \omega_i) I_i(\mathbf{x_i}, \omega_i), \cos\theta_i d\omega_i d\mathbf{x_i}, \tag{2.7}$$

where ρ is the BSSRDF computed from the (directional) dipole approximation.

If the scattering radius is very small, we can assume that $\mathbf{x_i} = \mathbf{x_o}$ and use Kubelka-Munk theory [23] to calculate an ideal diffuse appearance. Instead of integrating over the surface, the outgoing radiance is calculated using the ideal diffuse albedo a_{diff} [49]:

$$a_{diff} = 1 + \frac{K}{S} - \sqrt{\frac{K^2}{S^2} + 2\frac{K}{S}} \tag{2.8}$$

$$K = 2\sigma_a \tag{2.9}$$

$$S = \frac{3}{4}(1-g)\sigma_s \tag{2.10}$$

2.3.2 *Optical properties of teeth*

Enamel is a highly translucent material due to its crystalline prismatic structure and composition, with decreasing scattering coefficient at longer wavelengths. Dentin on the other hand consists of dentinal tubules, which are microscopic tubes that radiate outward from the pulp to the enamel. As a result of this structure, dentin is highly scattering, compared to the enamel. Finally, the pulp has very little influence on the appearance of the tooth. This fact is also known from root canal treatment, where replacing the necrotic tissue with zirkonia ceramic (normally used to replace damaged dentin) does not significantly change the color of the tooth.

The results of previous studies dealing with the optical properties of teeth are varying due to the measurement methods, the condition of the teeth and the used wavelengths. Furthermore, none of these studies contain a reference to the color of the used teeth with respect to a shade guide. We found the results of Zijp [49] most promising as he determined all of the required parameters for different wavelengths in the visible spectrum, where most others omitted blue in favor of near infrared. Since the wavelengths he used were not those of the sRGB primary colors, we had to interpolate/extrapolate some values. Note that absorption was not directly measured, but only the diffuse albedo, so scattering and absorption coefficients depend on each other. The results are shown in Table 2.1.

Table 2.1: Approximate optical parameters of dental tissues in mm^{-1}**.**

Dentin ($g = 0.44$)

σ_{aR}	σ_{aG}	σ_{aB}	σ_{sR}	σ_{sG}	σ_{sB}	η
0.65	0.60	0.75	120	125	130	1.63

Enamel ($g = 0.68$)

σ_{aR}	σ_{aG}	σ_{aB}	σ_{sR}	σ_{sG}	σ_{sB}	η
0.08	0.09	0.13	3.15	3.97	4.85	1.49

These values, however, are only taken from a single sample and do not capture the variation in tooth color. For monochromatic light, Zijp also measured the variation of scattering and albedo [49] showing that there is up to one order of magnitude between minimum and maximum. Due to the low absorption in the enamel, this parameter does not significantly contribute to individual differences. The dentin scattering only scales the albedo which can also be attributed to a change in the absorption parameter.

Since living teeth are wet and covered with saliva, we also have to take the refraction index of saliva ($\eta = 1.33$) in account. In addition, the necks of teeth are often covered with dental calculus and there

is some discoloring in fissures. These thin layers absorb a part of the incident and outgoing light.

2.4 DATA PREPARATION

The meshes of the restorations were generated by first fitting the deformable tooth models into a jaw that was scanned by a camera. This is done by first calculating a rigid transformation of a parametrized "average" tooth using the iterative closest point (ICP) algorithm [32] to calculate point correspondences. Then the deformable model of Blanz et al. [1] – i.e. a statistical shape model – is used to fit the enamel mesh to the tooth scanned before treatment. After this fitting process, a model of a restoration can be generated from a second scan where the carious material has been removed. This is done by clipping the mesh at the remaining tooth and locally deforming it to produce a smooth transition. Then the lower part of the restoration model, where it touches the remaining tooth, is generated. Note that the enamel mesh before clipping could also be used to render the original tooth when the algorithm is used for other purposes than a dental CAD system.

In addition to the restoration model we want to render, we use the tooth number to select the corresponding "average" tooth for each model. For each of these average teeth, we pre-generate a model of the dentin core once and store these on disk. At runtime we can then deform the average tooth and its dentin core to fit the tooth to be rendered as closely as possible using a non-linear space deformation (see Section 2.4.2 for details).

2.4.1 *Dentin core generation*

The distance between the tooth surface and the dentin core is up to 3 mm at the center of the cavity and linearly decreases to zero at the gum. For incisors, the distance between incisal edge and dentin core is only 2 mm, while the horizontal distance between enamel and dentin is up to 1 mm for both. This is due to the fact that the crown stage during which dentin and enamel are built is longer for molars (7–9 years) than for incisors (4–5 years). There is also a physiological reason since only the occlusal surface of molars is subject to abrasion. To generate the dentin model, we thus start with a model of the enamel surface and shrink it until the distance constraint (up to 1 mm horizontal and 2 or 3 mm vertical) is fulfilled. In contrast to Larsen et al. [26], we are however not limited to a simple deformation of the surrounding space because we only need to generate the dentin models for the average teeth once and then store them on disk.

For each vertex $\mathbf{v}$ of the dentin model, we gather all triangles T of the enamel model for which the distance to $\mathbf{v}$ is less than the enamel

$V :=$ set of all dentin vertices, $V_0 = V$
foreach iteration $i < i_{max}$:
  foreach vertex $\mathbf{v} \in V$:
    $T = \{\text{triangles } t : d(\mathbf{v}, t) < d_{max}(\mathbf{v})\}$
    $C = \{\mathbf{p} : \forall t \in T : d(\mathbf{p}, t) \geq d_{max}(\mathbf{v})\}$
    $\mathbf{v} = \frac{i}{i_{max}} P_C(\mathbf{v}) + (1 - \frac{i}{i_{max}})\mathbf{v}$
  foreach vertex $\mathbf{v} \in V$:
    $\mathbf{v}_1 = \frac{\sum_{w \in \text{neigbors}(\mathbf{v})} \frac{\mathbf{w}}{d(\mathbf{v}_0, \mathbf{w}_0)}}{\sum_{w \in \text{neigbors}(\mathbf{v})} \frac{1}{d(\mathbf{v}_0, \mathbf{w}_0)}}$
  $V = V_1$

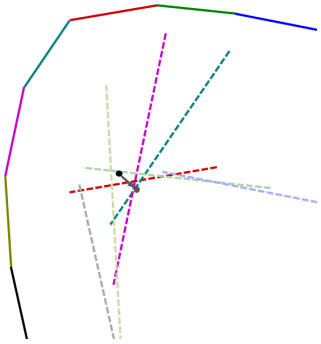


Figure 2.3: Dentin model generation: Pseudocode (left) and violated half space constraints and closest valid vertex position (right)

thickness $d_{max}(\mathbf{v})$. Each of these triangles defines a half space in which the vertex can be positioned. From the current vertex position, we calculate the closest point in the intersection of these half spaces and move the vertex there (see Figure 2.3). To prevent flipped triangles, we iterate this process several times and smooth the vertices between the iterations. During smoothing, we weight each neighbor vertex with the inverse of the original distance. In the first iteration we move the vertices only a fraction of the distance to the target point and linearly increase this factor to one.

This data generation stage needs about two to three minutes for each of the 11 average teeth models. Note that while humans have 4×8 teeth, all incisors and the canine teeth share a single average tooth model. In addition, only two pre-molars and three molars are required for both the lower and the upper jaw due to symmetry.

2.4.2 *Fitting of average teeth*

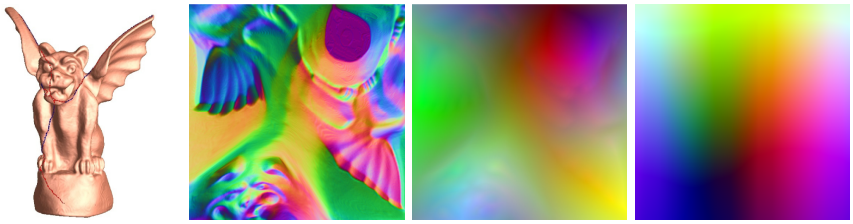


Figure 2.4: Geometry Images: Model and textures with normals and vertices from Gu et al. [15] and texture of tooth with vertices.

Since the average tooth and its dentin core have the same parametrization as the (partial) restoration (which originally comes from the deformable model), the corresponding vertices of the two models can be found by simply generating geometry images [15] for both. To deform the average tooth – and its dentin core – we compute tri-quadratic Bézier space deformation [34]. For each vertex $\mathbf{v}_i^{rest}$ of the restoration, we compute the matching template position $\mathbf{v}_i^{temp}$ and describe the de-

formation $\mathbf{X}(\mathbf{v})$ as an energy minimization problem that is regularized using the second derivatives of the control mesh.

$$E = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} \left\| \mathbf{X}(\mathbf{v}_i^{temp}) - \mathbf{v}_i^{rest} \right\| + \lambda E_{reg} \tag{2.11}$$

$$E_{reg} = \sum_{i,j,k} \mathbf{c}_{ijk}^{\partial^2 x^2} + \sum_{i,j,k} \mathbf{c}_{ijk}^{\partial x \partial y} + \ldots + \sum_{i,j,k} \mathbf{c}_{ijk}^{\partial^2 z^2}, \tag{2.12}$$

where $\mathbf{c}_{ijk}^{\partial a \partial b}$ are the control points of the Bézier volume describing the second derivative with respect to a and b, e.g. $\mathbf{c}_{000}^{\partial^2 x^2} = 6(\mathbf{c}_{000} - 2\mathbf{c}_{100} + \mathbf{c}_{200})$. This way, the deformed average tooth fits the restoration as closely as possible and is reasonably completed such that the scattering can be computed for a whole tooth. Figure 2.5 shows the fitted average tooth with its dentin core for an incisor crown. We chose $\lambda = 0.01$ for the regularization but it could be much lower if only complete restorations (or teeth) were rendered.

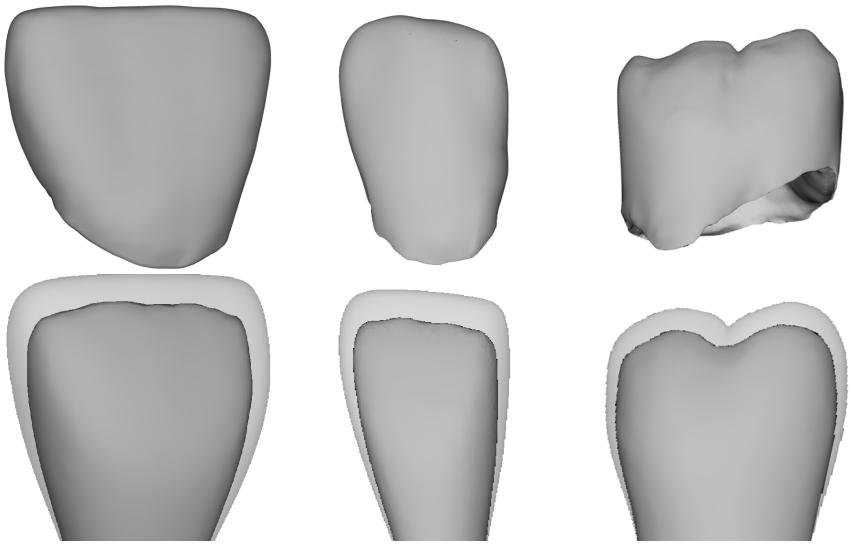


Figure 2.5: From left to right: Incisor, canine and molar restorations (top) with fitted corresponding average tooth and dentin core (bottom).

2.5 RUNTIME ALGORITHM

We use 5 render passes to render each tooth, plus additional passes for mipmap generation. Instead of using the model of the restoration to compute the subsurface scattering, we use the corresponding fitted average tooth and its pre-generated dentin core. The model of the restoration itself is only used in the last step. This way, we avoid artifacts when rendering only partial restorations like veneers or inlays.

In the intermediate steps, we do not render the models themselves, but only into their geometry image parameterization. The surface

mesh and normals are stored in a texture. For rendering, a regularly tessellated mesh is used and the position can be fetched from the texture per vertex. Levels of detail can be generated by simple mipmapping. This allows us not only to vary the quality and speed of the rendering, but also to compute the in- and out-scattering over the whole mesh.

2.5.1 *Rendering steps*

Our algorithm performs the following rendering steps:

1. Render gum and scanned jaw
2. For each restoration:
   a) Render the in-scattering in enamel from light source using the fitted average tooth.
   b) Compute scattering through enamel to dentin into geometry image of the dentin core.
   For each pixel: Sample in-scattering and position texture from a and calculate scattering through enamel.
   c) Scale dentin in-scattering by diffuse albedo and remap to out-scattering image.
   For each pixel: Store position, normal and outgoing radiance (scaled by local surface).
   d) Scattering through enamel and from dentin through enamel into geometry image of fitted average teeth.
   For each pixel: Sample in-scattering and position textures from a and c and calculate scattering through enamel.
   e) Render restoration with out-scattering – attenuated by fissures and dental calculus – and surface reflection from saliva.

In the first step, we render the tooth as seen from the light source into a texture map that stores the position, normal and texture coordinate of the enamel mesh. From these we can also calculate the incident light at each point in cd/m^2.

Similar to Fernández-Oliveras et al. [9], we approximate the distance dependent scattering profile due to multiple scattering using the effective anisotropy g_{dist}. This depends on the anisotropy constant g, the scattering coefficient σ_s and the material thickness d. We use a data driven approach in order to come up with a simple equation for the effective anisotropy. First, we have performed path tracing simulations (eqs. 2.1, 2.4 and 2.5) for enamel slabs with a thickness ranging from $0.1\sigma_{sr}$ to $100\sigma_{sr}$, where $\sigma_{sr} = (1-g)\sigma_s$ is the reduced

scattering coefficient. Then we calculate a least squares fit to the following equation:

$$g_{dist}(d) = \frac{2}{1 + a_1 d_{sr} + a_2 \sqrt{d_{sr}} + a_3 d_{sr}^{\frac{1}{3}}} - 1, \tag{2.13}$$

where $d_{sr} = d\sigma_{sr}$ is the thickness scaled by the reduced scattering coefficient. The fitting produced the following coefficients: $a_1 = 0.18$, $a_2 = 0.92$ and $a_3 = -0.52$. The effective anisotropy is shown in Figure 2.6 for enamel (red, green and blue light) and dentin. The maximum fitting error is below 0.01.

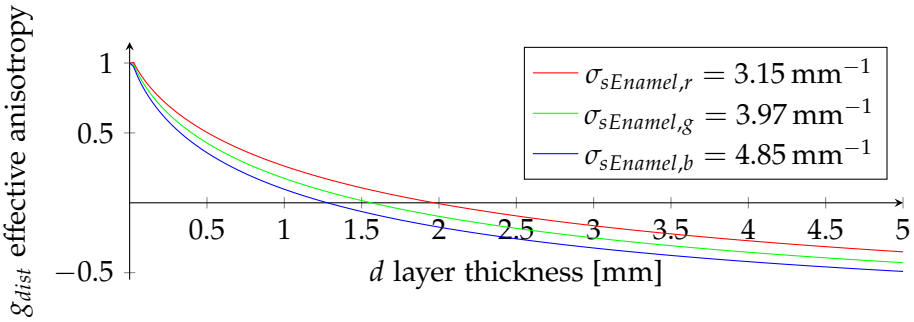


Figure 2.6: g_{dist} as a function of thickness in mm for average human enamel.

To approximate the scattering equation, we can now simplify the absorbing and scattering material as a single scattering sheet between two absorbing but non-scattering layers (see Figure 2.7). In contrast to previous work that only considers single scattering, we approximate multiple scattering events with this layer. The final diffuse out-scattering at $\mathbf{x_o}$, when the surface is orthogonally lit at $\mathbf{x_i}$ then becomes:

$$I_o(\mathbf{x_o}) = I_i(\mathbf{x_i}) F_t p_{dist}(\theta) e^{-\sigma_a(\|\mathbf{p_s}-\mathbf{x_i}\|+\|\mathbf{x_o}-\mathbf{p_s}\|)} \frac{\cos\theta_o}{\|\mathbf{x_o}-\mathbf{p_s}\|^2}, \tag{2.14}$$

where $\mathbf{p_s}$ is the scatter point at depth d_s, F_t the Fresnel transmittance, θ the scattering angle and θ_o the angle between surface normal and the vector $\mathbf{p_s} - \mathbf{x_o}$.

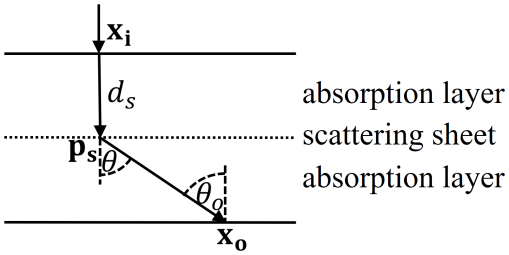


Figure 2.7: Geometric configuration for light scattering through the enamel layer.

We also need to determine the effective scattering depth d_s in addition to the effective anisotropy. To guarantee symmetry of the geometric configuration for in- and out-scattering through enamel, we use $d_s = 0.5d$. In the general case, where the incident light direction is not tangential to the surface, we have:

$$\mathbf{p_s} = \mathbf{x_i} + \frac{d_s}{\cos\theta_i}\mathbf{L_i}, \tag{2.15}$$

where $\mathbf{L_i}$ is the refracted incident light direction. Note this means that we only consider either the incident or outgoing light direction, if we swap $\mathbf{x_i}$ and $\mathbf{x_o}$. Figure 2.8 shows the resulting scattering profiles compared to path tracing for the range of up to a few millimeters.

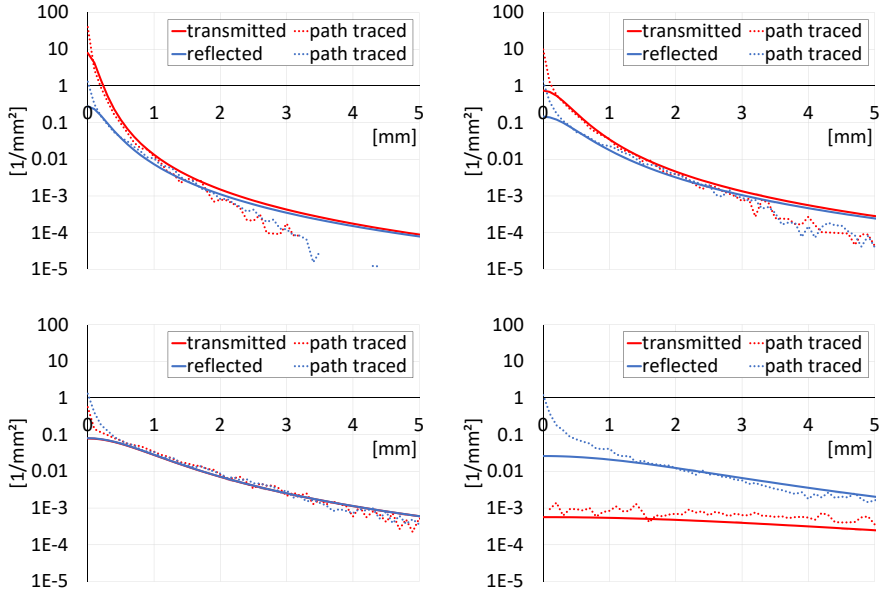


Figure 2.8: Enamel scattering profiles for 0.5, 1, 2, and 5 mm (from left to right and top to bottom), for red light (633 nm). For other wavelengths, all distances are scaled accordingly.

We store the in-scattering through the enamel onto the dentin surface by integrating the incident light from the dentin using equation 2.2 with the Henyey-Greenstein phase function (eq. 2.1) and our modified scattering coefficient (eq. 2.13):

$$I_o(\mathbf{x_o}) = \int_S I_i(\mathbf{x_i}) p(\theta) e^{-\sigma_a \|\mathbf{x_i}-\mathbf{p_s}\|} \frac{\cos\theta}{\|\mathbf{x_i}-\mathbf{p_s}\|^2} d\mathbf{x_i} \tag{2.16}$$

$$p(\theta) = \frac{1}{4\pi} \frac{1-g_{dist}^2}{(1+g_{dist}^2-2g_{dist}\cos(\theta))^{1.5}} \tag{2.17}$$

$$g_{dist} = \frac{2}{1+0.18d_s+0.92\sqrt{d_s}-0.52d_s^{\frac{1}{3}}} - 1 \tag{2.18}$$

$$d_s = d\sigma_s(1-g), \tag{2.19}$$

where $\mathbf{x_i}$ is the sample point on the enamel surface, $\mathbf{x_o}$ the point on the dentin surface and θ the angle between the incident light direction – calculated using the law of refraction and the vector from the light source to $\mathbf{x_i}$ – and the outgoing light direction, i.e. $\mathbf{x_o} - \mathbf{x_i}$.

To sample the incident light on the enamel, we use the pattern shown in Figure 2.9 (left). Compared to the 21-tap [6] or 12-tap sampling, used in other approaches [25, 26, 36], it covers a wider area and takes a total of 64 instead of 21 or 12 samples. This is necessary since the absorption in enamel is almost zero and scattering varies strongly with distance. The smallest samples cover a single pixel for both sampling patterns, so we cover 64×64 pixels while the 21-tap kernel only covers 15×15 pixels. Again, we store the incident radiance. Storing the direction is not necessary due to the strong scattering in the dentin core. The sampling kernel is centered at the point on the enamel surface from which the incident light is directly refracted into the shaded point on the dentin core. This point is found by projecting the shaded point on the enamel surface and then performing an iterative search using steepest decent. The derivative of the distance between the refracted ray and the shaded point is analytically calculated from the normal at the enamel surface point. We also tested using the point on the enamel surface with the same parameters as starting point but this did not reduce the required number of iterations. In both cases, a sufficient accuracy is reached after 20 iterations.

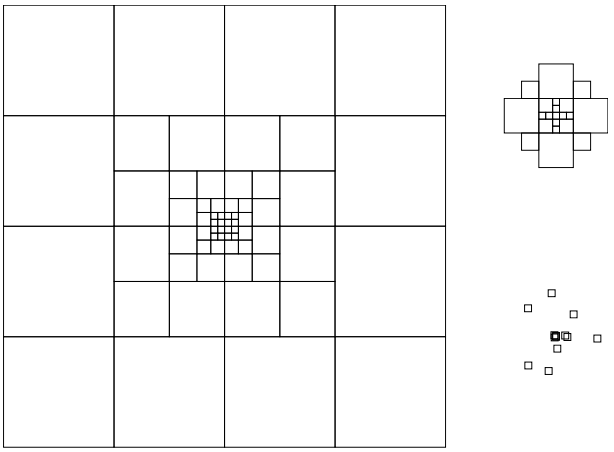


Figure 2.9: Our enamel sampling pattern (left) compared to the 21-tap (top right) and 12-tap (bottom right) kernel. The size of squares indicates the used mipmap level.

Due to the high scattering coefficient of dentin, the dentin core appears almost ideally diffuse. Even with the lowest scattering and absorption parameters measured in human dentine, the scattering radius is less than 0.1 mm. This effect is increased by the fact that incident and outgoing radiance are smoothed by the scattering inside the enamel. As both incident and outgoing light are very smooth over the dentin surface, we can use low resolution textures, i.e. up

to at most 64×64 pixels for both. Again, we store the incident light (now into the enamel layer) in the out-scattering texture. Storing the direction is again not necessary since the out-scattering is diffuse. The scattering could be calculated using the classical dipole diffusion model [22] together with the 21-tap kernel of Dachsbacher et al. [6] because of the high scattering and absorption coefficients. Due to the almost diffuse appearance, we can however also use Kubelka-Munk theory [23] to calculate an ideal diffuse appearance and calculate the outgoing radiance using ideal diffuse albedo a_{diff} instead (eq. 2.8).

For the final out-scattering from the enamel, we again use the parameterization of the enamel mesh to store the emitted radiance into view direction. Again, we compute the scattering integral (eq. 2.16) and store the outgoing radiance in cd/m^2 using our 64-tap kernel. For this pass we swap $\mathbf{x_i}$ and $\mathbf{x_o}$ to calculate p_s because the dentin out-scattering is almost diffuse but the out-scattering from enamel depends on the view direction. This means that $\mathbf{p_s}$ actually depends on $\mathbf{x_i}$ and is not constant for the whole integral any more. To compute the integral, we first remap the dentin out-scattering texture to an image similar to the enamel light map. Since surfaces facing slightly away from the viewer can also contribute to the scattering, we use an inverted view frustum and an inverted depth test. The field of view is 90 degrees and the distance of the camera to the center of the dentin mesh is the mesh diagonal. Figure 2.10 shows the geometric configuration.

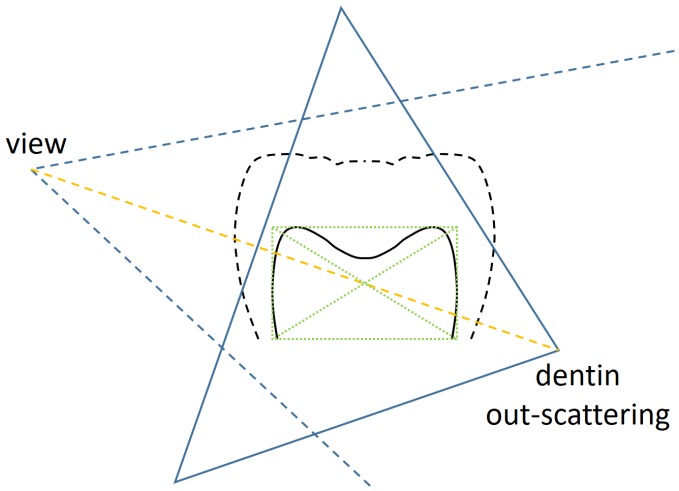


Figure 2.10: Geometric construction of the inverted view frustum (solid blue) used to generate the remapping texture for the dentin out-scattering.

In contrast to the in-scattering through the enamel layer, searching for the center point of the sampling pattern can be performed using a simple bisection search along the refracted eye ray. Here we only use 10 iterations and set the initial search interval to the diameter of a tooth, i.e. $[0..1]$ cm, leading to an accuracy of 0.01 mm.

Figure 2.11 shows the intermediate textures generated by our rendering algorithm and the final result with out-scattering mapped to the tooth and additional highlights. Note the smooth appearance of the two dentin textures.

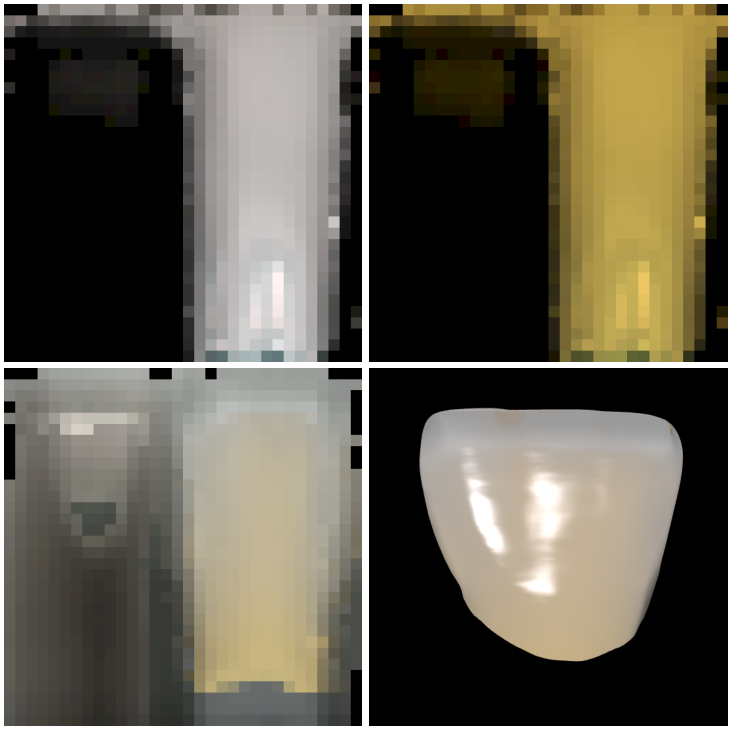


Figure 2.11: The results of the rendering passes b-d: In scattering into dentin (top left), out-scattering from dentin (top right), out-scattering from surface (bottom left) and final result (bottom right). The light map (step a) was omitted.

2.5.2 *Surface structure*

Finally, we add the remaining details to the surface after mapping the out-scattering to the model of the restoration. To simulate the wet saliva, we add a specular highlight using a cosine lobe with an exponent of 50. The amount of reflected light is calculated from the index of refraction $n = 1.33$ scaled by the Fresnel factor estimated using the approximation of Schlick [33]. In addition, we use procedural bump mapping to simulate the perikymata by perturbing the surface normal for the specular reflection with Perlin noise [28]. Furthermore, we estimate the position of fissures by calculating the maximum negative curvature of the mesh in the region of the tooth crown. The fissure intensity is then computed by simply scaling this value and using it to attenuate the out-scattering. The parameters, e.g. frequencies and amplitude of the noise and minimal negative curvature and inten-

sity of fissures, can be modified at runtime and default values were determined by a dental technician.

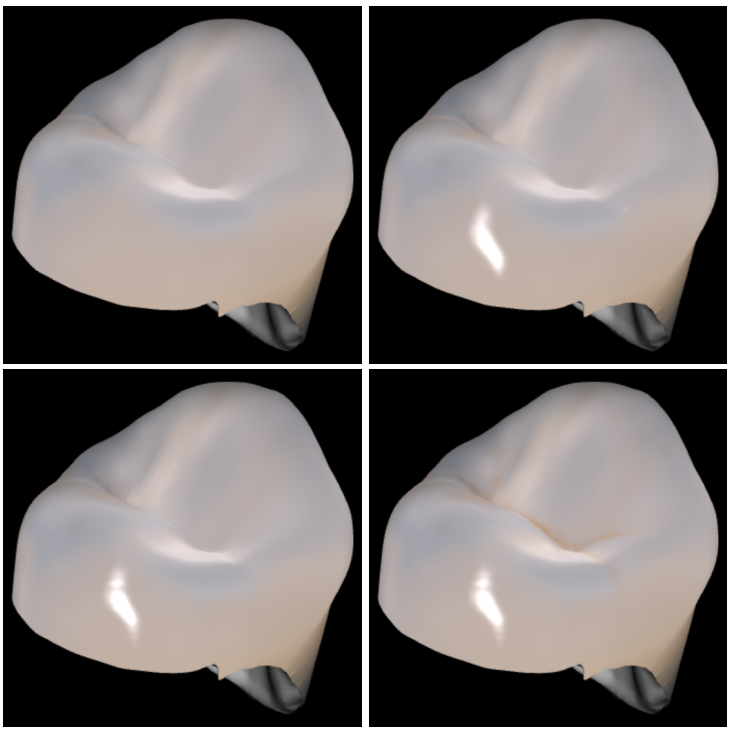


Figure 2.12: Results of the different surface structure effects on a molar partial crown (shade: 2M2): Without surface effect (top left), saliva only (top right), saliva and perikymata (bottom left) and all inluding fissure (bottom right).

Figure 2.12 shows the different effects of the surface structure. Note that the effects of saliva and perikymata are also present in the shade guides (cf. the jittered reflections in Figure **??**), but fissures of course not as they only appear on molars.

2.6 SCATTERING PARAMETERS

As mentioned above, the scattering parameters of teeth vary strongly between humans leading to different tooth colors. This fact has been considered in restorative dentistry for several decades. Special "shade guides" are used to determine the color of the tooth to be treated or replaced (see Figure 2.18 in the results section).

Only few measurements of the scattering parameters have been made in literature and none of them were related to tooth shades. As the tooth shade has subtle effects on the appearance of a tooth, it is however an important factor when modeling the shape of a restoration. To reproduce the exact tooth shade, we need to estimate the scattering parameters for each of them in the used shade guide.

2.6.1 *Appearance capturing*

The exact color of the shade guide samples is also required to build automatic color identification tools. These combine controlled illumination with a photo sensor to measure the reflectivity of the sample. For such tools, the colors of the shade guide samples were measured, e.g. by Kuo [24]. These measurement however only consist of a single CIE-Lab color value sampled at the center of the tooth.

To reproduce the overall appearance of a tooth, we first observe that there is a typical color gradient from the incisal edge over the center to the neck of the tooth. We therefore took images of the two VITA shade guides (Figure 2.18) under sunlight at a clear sky and measured the color at the incisal edge and the tooth center by averaging an area of 5×5 pixel in the image. To measure the scattering only, we avoid the reflection by using a slight offset between the sun and the camera. Using the data from Kuo [24], we corrected the color of the central point and applied the same scaling in RGB to the incisal edge as well.

2.6.2 *Parameter fitting*

Although the two important tooth layers enamel and dentin have four scattering parameters in total, the appearance of the tooth is mainly determined by only two of them. Most of the color variation at the incisal edge comes from the enamel scattering constant which in general decreases with age, i.e. demineralization of the tooth. The second important constant is the light absorption in the dentin layer that mainly determines the color of the tooth center. Varying the light absorption in enamel within the range of values found in human teeth does not produce any significant visual differences. Due to the high scattering coefficient in dentin, it appears almost diffuse and therefore the appearance depends on the ratio of absorption to scattering. We therefore can fix one – i.e. the scattering coefficient – and only estimate the other one.

Starting with the color of the two sample points on the tooth, we use the Levenberg-Marquard algorithm [27] to determine the parameters mentioned above. The cost function is the L^2 difference of the colors at the two sample points in the CIE-Lab color space. The function is computed by rendering an incisor model – lit and viewed from the front – and again averaging the color of a 5×5 pixel block at the center of the incisal edge, where the material is solely enamel, and a block close to the neck of the tooth, where only a thin enamel layer is in front of the dentin core. During the fitting procedure, we switch off rendering of the saliva reflection to only account for the scattering as mentioned above for the photographs. The optimization requires approximately 100 iterations and needs less than one second for each shade.

Note that we currently do not directly consider back lighting in the fitting procedure, although this could simply be integrated by capturing and rendering a second view. Since teeth are however almost always lit and seen from the front, this does not impair rendering quality in typical applications. In the area of dental CAD, the light source is even always at the view position since this is a "natural" setup for dentists and dental technicians.

2.7 RESULTS

As we made several assumptions and simplifications in the rendering process, we first visually compare the generated images with a path traced solution. Figure 2.13 shows a comparison to an image generated using a volumetric path tracer. We again simulate the photon paths using eqs. 2.1, 2.4 and 2.5. We have removed the surface effects for both images as these are identical anyways. Beside the slight color shift and the stronger blurring of the dentine at the incisal edge, the scattering looks similar. While the color shift is compensated by our parameter fitting procedure, this might lead to slightly incorrect scattering parameters when used for other rendering methods. The blurring of the incisal edge is mainly due to low resolution of the out-scattering texture.

Figure 2.13: Incisor out-scattering rendered with our method (left) compared to a path traced image (middle) and difference between them (scaled by 4, right). Both use the same scattering parameters from Zijp [Zij01].

Figure 2.14 shows a comparison to the approach of Larsen et al. [26]. While their renderings also reproduce some color variation compared to a diffuse BRDF due to subsurface structures, the overall appearance is rather diffuse and the translucency of the enamel layer is too little because of neglecting multiple scattering. Our algorithm more faithfully reproduces the shade and appearance of a real tooth in real-time.

The main goal of our approach was to develop a real-time rendering algorithm for teeth that is able to reproduce the different shades of human teeth and scales to the power of the graphics hardware. We therefore evaluate the frame rate with respect to different texture

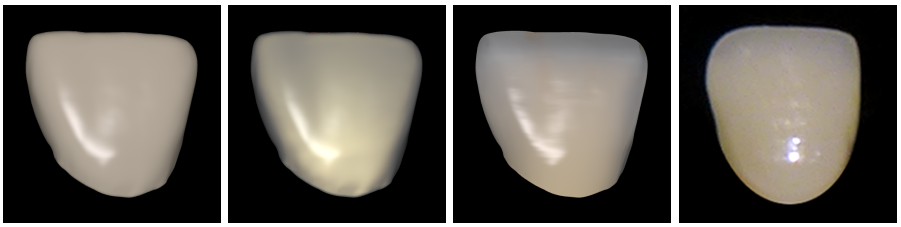


Figure 2.14: From left to right: Incisor rendered with diffuse BRDF, with the approach of Larsen et al. [LFJB12] and our proposed method. The rightmost image is photo of a veneer shown for reference.

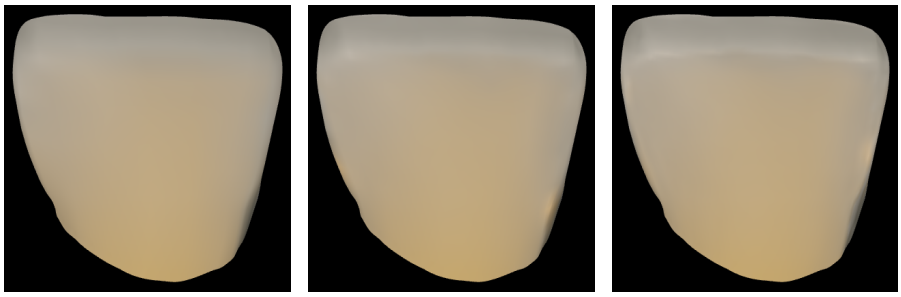
resolutions. Table 2.2 shows the frame rate for up to 14 teeth on a Geforce GTX 1080 graphics card. For resolutions up to 32×32, the algorithm is purely limited by the number of transformed vertices and rendering passes. It becomes pixel shader limited at a resolution of 64×64 pixel for the in- and out-scattering textures. For a full set of 32 teeth, we would achieve approximately 40 frames per second at a resolution of 32×32 pixel. Note that these timings are for a dynamic scene, i.e. in- and out-scattering are calculated for every frame.

Table 2.2: Frame rate and timings measured with Geforce GTX 1080.

texture res.	16×16	32×32	64×64
Incisor	$800(1.2ms)$	$761(1.3ms)$	$571(1.8ms)$
Veneer (2 teeth)	$523(1.9ms)$	$500(2.0ms)$	$380(2.6ms)$
Bridge (6 teeth)	$222(4.5ms)$	$200(5.0ms)$	$150(6.6ms)$
upper jaw (14 teeth)	$95(10.5ms)$	$91(11.0ms)$	$66(15.2ms)$

Each average tooth and corresponding dentin core is composed of 5k to 10k triangles and the restorations contain up to 50k triangles. Since the restorations and other models like gum are rendered only once with simple texture lookups and low complexity shaders, our approach scales mostly with the texture resolution, as can be also seen in Table 2.2. As a lower texture resolution reduces the accuracy of the scattering within the dentin core, we visually compare the result for different resolutions in Figure 2.15. While the difference between 32×32 and 64×64 is almost negligible, the visual quality degrades when reducing the resolution to 16×16 mainly due to detail loss in the enamel out-scattering.

Table 2.3 shows the range of absorption and scattering values determined by our fitting algorithm. The values are within the range determined by previous work on measuring the optical parameters of teeth. The only exception is $\sigma_{aDentin}$ where we have determined higher values for blue. The main reason is that we combine dentine and pulp, where the hemoglobin in the pulp exhibits strong absorption



(a) Resolution 16×16 **(b) Resolution** 32×32 **(c) Resolution** 64×64

Figure 2.15: Quality of enamel out-scattering for different texture resolutions (shade 3R2.5).

at short wavelengths. So the absorption parameters can be seen as a combination of dentin and hemoglobin.

Figures 2.16 and 2.17 show how well our method works for partial restorations. Although only parts of the teeth are rendered, the scattering is computed for the complete teeth. This allows generating renderings that show how a restoration will look when it is attached to the tooth. Actual restorations are often layered onto an opaque base that is rendered diffuse gray in our system.

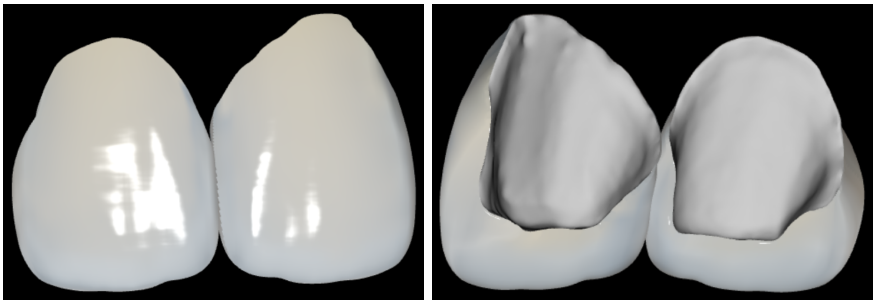


Figure 2.16: Rendering results of veneer restorations (shade 0M1).

As shown in Figure **??** we can accurately reproduce the shade guide samples after parameter matching. Figure 2.19 shows all shades of the two evaluated shade guides, cf. Figure 2.18. Note that our renderings show a slightly stronger "opalescence" at the incisal edge (blueish tint) than the shade guides but this is also stronger in real teeth than in the shade guides.

Table 2.3: Range of scattering and absorption values determined by our fitting procedure.

	R	G	B
$\sigma_{sEnamel}$	7.463 – 16.14	7.731 – 17.09	7.141 – 22.43
$\sigma_{sDentin}$	30.00 – 170.5	31.25 – 177.5	32.50 – 184.6
$\sigma_{aDentin}$	0.425 – 1.734	0.800 – 10.61	5.607 – 50.00

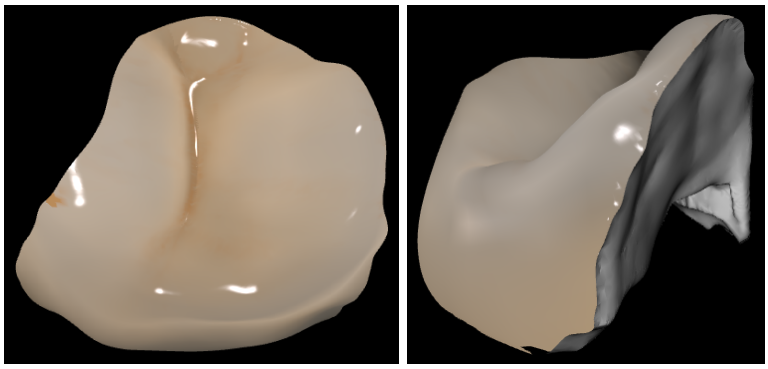


Figure 2.17: Rendering results of an inlay restoration (shade 2L2.5).

Figure 2.18: VITA classical (top) and 3D-Master (bottom) shade guides to determine tooth color.

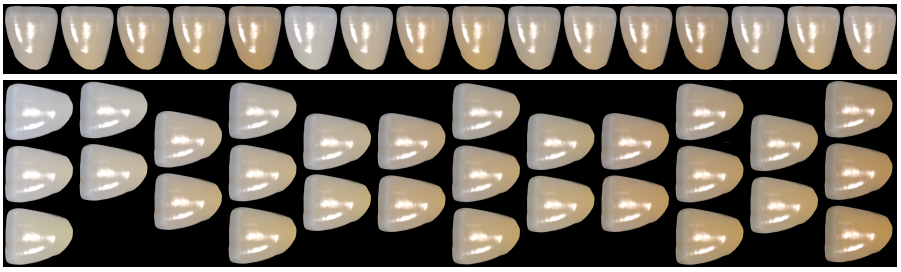


Figure 2.19: Rendered incisors showing all colors of the two evaluated shade guides: VITA classical (top) and 3D-Master (bottom).

Finally, we visually compare our rendered teeth to photographs with the same tooth shades (see Figure 2.20). Note that the mouth and gum are not rendered but copied from the photographs since this is not the scope of this work.

2.8 PRECOMPUTED TEXTURE ATLAS

In order to reduce the rendering times, we generated a texture atlas for the out-scattering from surface (rendering step d) by calculating the results for multiple uniformly distributed positions on a sphere around the restoration for both camera and light. As tests showed, 92 (geodesic subdivided icosahedron) positions are sufficient for generating interpolated textures that resemble the correct result. A 64×64 texture

Figure 2.20: Rendered teeth (left) compared to actual photographs (right); mouth and gum are taken from the photos. Note that the geometry of the teeth does not match those of the subjects. The tooth shades are 2M3 (top) and 4L2.5 (bottom). Artifacts on the rightmost molar are due to broken texture coordinates at seams.

resolution therefore results in a 5888×5888 texture atlas. Assuming realistic use cases, the possible positions can by greatly reduced.

2.9 CONCLUSION AND PROSPECTS

The proposed technique is able to render dental restorations and whole human teeth in real-time with varying positions of light and viewer. By fitting the scattering parameters to different shade guides, we can also reproduce the correct tooth color.

The main limitation is that we can only render dental restorations or sound teeth. This is due to the fact that we compute in- and out-scattering for a complete (sound) tooth and use a fixed dual-layer structure. Thus, we are especially not able to render broken teeth with dentin or pulp at the surface. Another limitation is that we assume the background to be black or at least sufficiently dark to neglect translucency. This is, however, a reasonable assumption for teeth inside the mouth.

Due to the limited size of the out-scattering texture and the missing peak of the scattering profiles for very thin enamel layers, shadows are also blurred slightly too much close to the neck of the tooth.

3

PATHFINDING BASED ON LANDMARKS

3.1 MOTIVATION

In the last decades, there have been multiple approaches to reduce the complexity of path finding algorithms like Dijkstra and A* [18], either by exploiting the structure of the graph, by pre-computing results or by accepting non-optimal results. The most commonly used algorithms for uniform grids in this area are Hierarchical Path-Finding A* (HPA*) [2] alongside with Subgoal Graphs [42] and jump point search (JPS) [17] with its optimized variants. For more generalized graphs, the popular choices are hierarchical approaches or methods based on pre-computed minimal distance fields with multiple notation like landmarks [13] as well as differential heuristics [8]. Landmarks are randomly or intentionally chosen nodes of a graph. In a distance field associated with a landmark, each node contains the minimal distance to that landmark. Other approaches try to use the GPU for path finding in order to capitalize on the parallel processing capabilities of its architecture. Among these are either completely new parallel algorithms, beginning with March of the Froblins [37], adaptions of originally sequential algorithms, for example iterative deepening A* (IDA*) [20] or GPU implementations of A* itself, like GPU-A* [47].

In contrast to most other approaches, we use the distance fields directly for path finding instead of only as a heuristic for A* or other algorithms. Therefore, only two stacks containing the paths to the current search states are required for our bidirectional search. No open list or sorting is needed, making the runtime always linear in the number of visited nodes for which examples are shown in Figure 3.1. Additionally, we kept our algorithm as generic as possible so that it is not limited to grids or planar graphs but can also handle 3D structures like meshes. For simple structures like grids, it can be optimized to further reduce the non-optimality. Finally, dynamic environments are supported as well as disjoint graphs. Only directed graphs must be treated differently due to the traversal algorithm being tailored towards their undirected counterparts.

3.2 RELATED WORK

Apart from our previous work [30] which we extend here, there have been multiple approaches using distance fields for path finding. However, to our knowledge, they either use the fields for direct routing over landmarks [4] or as heuristic [13] for A* variants. While landmarks as

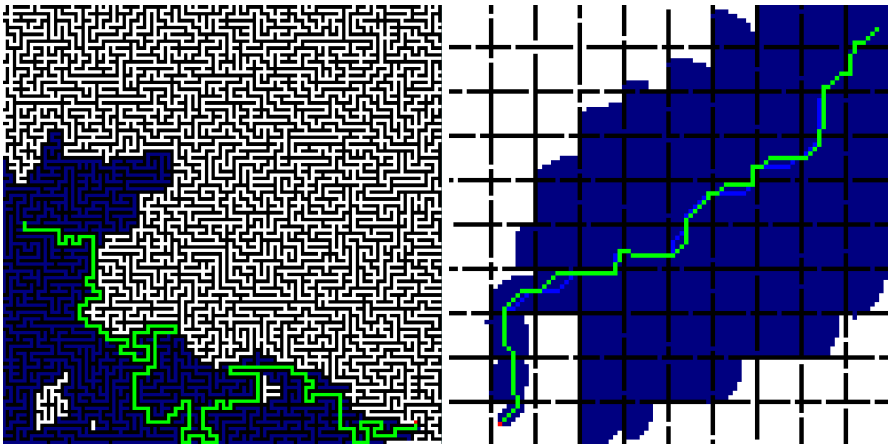


Figure 3.1: A* and its expanded nodes (blue) compared to our method (green). Using the euclidean distance as heuristic, A* visits a substantial percentage of nodes, while our approach directly finds a path.

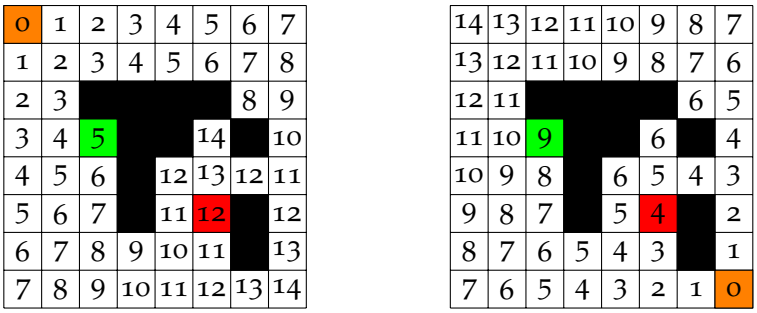


Figure 3.2: Example for landmarks (orange) and their distance fields. The lower bound for the highlighted nodes (red and green) is 7 (left field), the upper bound is 13 (right field).

heuristic can greatly reduce the number of visited nodes for A*, the main issue is still the requirement of a sorted list for traversal. The most closely related approach to our algorithm is LPI [14], which also routes over landmarks, but tries to combine the path over the nearest landmark to the start location with the path over the nearest landmark to the destination by finding an intersection. This leads to paths that are far longer than the optimal solution and requires smoothing to produce reasonable paths. Other approaches, mainly in the area of street navigation, use hierarchies [12] to compress the graph but they cannot be generalized easily.

3.3 BACKGROUND

Given a set of landmarks $L = \{l_1 \dots l_n\}$ with their distance fields $F = \{f_1 \dots f_n\}$ and let $f_i(w)$ be the distance value of the field i for the node w, then the lower bound d_{min} for distance $d(u, v)$ between two

nodes u and v can be calculated as follows according to Goldberg and Harrelson [13]:

$$d_{min}(u,v) \quad \geq \quad max_{1\leq i\leq n} |f_i(v) - f_i(u)| \tag{3.1}$$

Similarly, an upper bound d_{max} can also be calculated from $f_i(u)$ and $f_i(v)$ by

$$d_{max}(u,v) \quad \leq \quad min_{1\leq i\leq n} |f_i(v) + f_i(u)| \tag{3.2}$$

since both u and v can be reached from a landmark l if they are both defined in f. If either entry in f for u or v is undefined, there is no path between the two nodes. Figure 3.2 shows examples for landmarks and their distance fields including d_{min} and d_{max}.

Let $\varepsilon(u,v)$ be the maximum difference to the optimal solution that can occur while calculating the path from u to v then the following holds:

$$\varepsilon(u,v) \quad \leq \quad 2min_{1\leq i\leq n}(min(f_i(v), f_i(u))) \tag{3.3}$$

This is due to the fact that in the worst case with respect to Equation (3.2) a path from u to v going through f is constructed, which contains the complete path and reverse path to either u or v respectively.

This property can also be used as a quality criterion for the distribution of the landmarks L. Since the maximum path length depends on the largest distance to a landmark, it is reasonable to place additional landmarks at maximum distance to existing ones.

3.4 ALGORITHM

3.4.1 *Pre-processing*

At least one distance field needs to be constructed in the pre-processing stage for initialization. Additional fields can be generated later during run time if required. For directed graphs, we require twice the number of fields as a second field must be generated in the reverse direction of the graph. Details are described in Section 3.4.5.

The first landmark can be placed randomly, but we chose to select one outer node of the graph as our tests have shown that this results in a favorable distribution. This has also been pointed out by Goldberg and Harrelson [13] for street networks. In the case of unknown topologies, we cannot use this heuristic and simply choose a random node.

For disjoint graphs, we enumerate each graph with an index and store it in a separate index map. If there is a set of nodes that was not visited during the generation process of the distance field and therefore belongs to other graphs, we choose the node with the nearest euclidean distance to the current landmark and generate a distance field starting

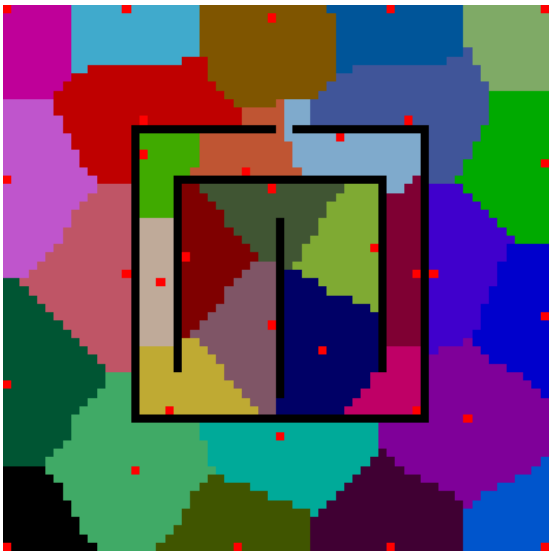


Figure 3.3: Voronoi like map, showing the landmarks in red and their nearest nodes.

from that node. Again, in case of graphs with unknown topologies we resort to choosing a random one. This process is repeated until each node was visited.

Additional landmarks for each graph can be placed either randomly or successively at the node with the highest distance to all previous landmarks. These nodes can be found using an additional map that stores the minimal distance to the closest landmark, which results in a Voronoi-like distance map as shown in Figure 3.3.

Using this approach, we construct a set of distance fields for a set of graphs, together with two additional maps, one for the minimal distance and one to specify the graph the node belongs to. For j nodes and n distance fields this results in $\mathcal{O}((j+2)*n)$ memory consumption.

3.4.2 *Field selection heuristic*

A critical step of the algorithm is choosing the most suitable field. Therefore we use the following heuristics. For the start and target nodes s and t we select the field f_i that maximizes $|f_i(s) - f_i(t)|$, which means that we favor the one resulting in the tightest lower bound $d_{min}(s,t)$. If that fails because $f_i(s) = f_i(t)$ for all i we choose the field that minimizes $min(f_i(s), f_i(t))$, i.e. the one that belongs to the landmark closest to s and t.

3.4.3 *Pathfinding*

The basic idea of our approach as described in Algorithm 1 is to follow a distance field to its origin or landmark beginning at the given start

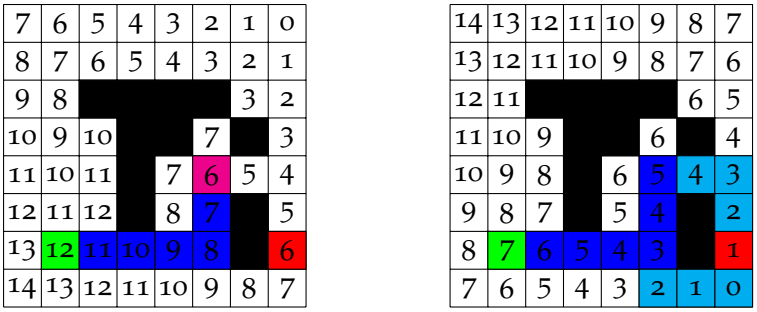


Figure 3.4: Examples for artifacts: on the left the previously found path (blue) at the current state (magenta) that requires a field change. On the right two probable paths, resulting in either a path that is too long (top cyan) or a loop back down(bottom cyan).

node, ideally finding the target node along the way. Therefore we have to select an appropriate field using our heuristic as proposed in Section 3.4.2, which hereinafter we refer to as the "best" distance field. Depending on the selected field, we either walk from the original start to the target or have to swap them if the field value of the start node is less than the value of the target node. Therefore we have two stacks, *path* and *reversepath*, which contain the path found so far from the original start and target to the current states as shown in Figure 3.5.

Figure 3.5: Illustration of the two node stacks *path* and *reversepath*, starting at s and t and resulting in a full path when $s_{curr} = t_{curr}$.

In order to follow the distance field, we search for a potential parent of each node by iterating over all neighbors. From these we pick a node that satisfies the condition that its field value is equal to the current field value minus the edge cost. Then the current (start) node s_{curr} is pushed on the active stack, the edge cost is added to the current path cost and the parent replaces the current node. If the current start and target nodes match, a path was found otherwise, the algorithm continues.

Since the original chosen field is not necessarily the best field, we check several times if a better field can be found. There are two triggers for swapping the field: either the nearest landmark has changed according to the minimal distance map as illustrated in Figure 3.3 or the field value of the current start node is equal or smaller than the value of the target node. If no usable field is found, the algorithm returns false as it cannot find a path between the nodes.

Due to the field changes, artifacts may occur as shown in Figure 3.4. Simpler artifacts can be found and removed by checking if the current node is already on the stack and removing the loop. Since this also consumes memory bandwidth, we only check the two previous nodes.

If the path cost is higher than the upper bound $d_{max}(s,t)$ the algorithm also returns false since routing over the landmark would result in a shorter path.

Algorithm 1 Descent algorithm

```
1:  function findPath( )
2:      if field(t_curr) − field(s_curr) > 0 then
3:          swap path and reversepath
4:          swap s_curr and t_curr
5:      while cost < d_max(s,t) do
6:          for all neighbors of s_curr do
7:              find parent
8:          if loop found then
9:              remove loop nodes
10:         else
11:             add parent to path
12:             replace s_curr with parent
13:             add edge cost to cost
14:         if s_curr = t_curr then
15:             return true
16:         if field change needed then
17:             find better field
18:             if field(t_curr) − field(s_curr) > 0 then
19:                 swap path and reversepath
20:                 swap s_curr and t_curr
21:             if no suitable field found then
22:                 return false
23:     return false
```

The complete approach consisting of several steps is shown in Algorithm 2.

These are mainly for preparation, merging and handling cases in which the base approach fails. Therefore we check at the beginning via the graph map if the target node is reachable from the start node. Then the best field mentioned above is chosen.

In order to speed up the process, the algorithm checks if it is possible and efficient to reach the target node by iteratively choosing the neighbor node that is closest in respect of the direction of the target node. This is essential for grid maps with large open areas or areas with only few connections, because the field values do not differ significantly in this case. It is also tried if Algorithm 1 failed. During the next step we search for a new valid field to retry Algorithm 1 with the current start and target nodes. This is repeated until the cost is higher than the upper bound $d_{max}(s,t)$ or a path is found. If no valid path can be found or the path costs exceed the upper bound $d_{max}(s,t)$, we fall back to the field that minimizes $d_{max}(s,t)$ and use the paths

that lead from the start and target nodes to the landmark of the field. The final step is to merge both stacks, removing all duplicate nodes that may be present on top of the stacks because of field changes or the $d_{max}(s,t)$ method that routes over the landmark.

Algorithm 2 Pathfinding algorithm

```
1:  function getPath(s, t)
2:      new variables s_curr = s and t_curr = t
3:      if t not reachable from s then
4:          return emptypath
5:      new stacks path and reversepath
6:      find best field field and d_max(s, t)
7:      if found direct path and pathlength < d_max(s, t) then
8:          return path
9:      while cost < d_max(s, t) and not findPath( ) do
10:         if direct path found between t_curr and s_curr then
11:             break
12:         try to find new field
13:         if no field found then
14:             break
15:     if cost > d_max(s, t) or the ends of the do not match then
16:         return d_max(s, t) based path
17:     merge path and reversepath
18:     return path
```

3.4.4 *Dynamic environments*

For dynamic environments, partial recalculation of the distance fields is required. The worst case is splitting or joining graphs, where the least optimal time is $\mathcal{O}(n/2)$ based on the distribution of the fields. Since our solution provides already reasonable results for a small number of fields, the recalculation can be done on a field by field basis, reducing the required update time.

3.4.5 *Directed graphs*

Directed graphs must be treated separately, since path finding is based on backtracking and the construction of distance fields proceeds along the directed edges. For this reason, only distance fields where the start node u has a lower distance field value than the end node v can be used. Furthermore, it is not possible to determine the upper bound $d_{max}(u,v)$ this way, since it is not guaranteed that a path from u to the landmark and thus to v is found.

In order to solve these problems, the inverse of the graph is used to generate an inverse field for each distance field beginning at the same landmark. For $f_{inv,i}$, the field inverse to field f_i, the upper bound $d_{max}(s,t)$ is given by:

$$d_{max}(s,t) \quad \leq \quad min_{1 \leq i \leq n} |f_{inv,i}(s) + f_i(t)| \tag{3.4}$$

The path finding is illustrated in Figure 3.6. The general algorithm $d_{min}(u,v)$ is used for field selection but it is necessary to distinguish between normal and inverse fields: if a normal field is selected, the *reversepath* is filled and accordingly for an inverse field the *path*.

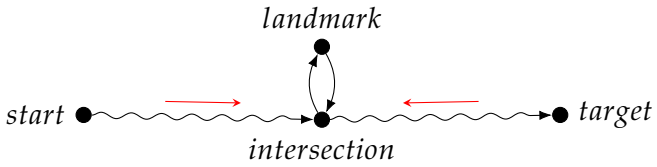


Figure 3.6: Direction of backtracking and real direction for directed graphs. The snake arrow denotes a permissible path, the straight red arrow the backtracking direction. The *intersection* node can be identical with the *landmark*.

3.5 RESULTS

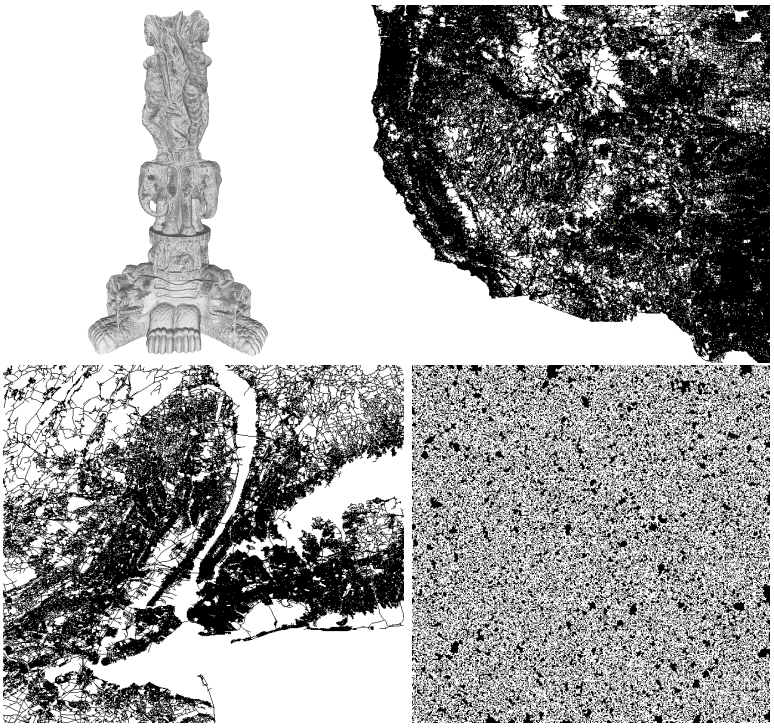


Figure 3.7: Used structures: Thai Statue, West Coast, New York and the random512-35 map (from top left to bottom right).

For the benchmarks we used 100 landmarks and distance fields on average. Our experiments have shown that this number yields good results while using reasonable amounts of memory, making it a good tradeoff between performance and accuracy. The landmarks are placed at the nodes with the highest distance to all other landmarks for reproducibility. Random placement is also discussed at the end of the section.

3.5.1 *Hardware and software*

Unless otherwise denoted, we use an Intel i7-6700K with 16GB DDR3 RAM and as GPUs the Radeon RX480 and RX5700. For the implementation we use C++ and OpenGL for the compute shaders in order to achieve platform independence. The GPU implementation is similar to the CPU implementation apart from transferring the path data to the client. For the distance field generation on the GPU we use OpenCL 1.2 as a common basis, although OpenCL 2.0 or CUDA could possibly improve the pre-processing performance.

3.5.2 *Suboptimality and length ratio*

As quality criterion, we use the length ratio which is the mean length of our approach divided by the mean optimal path length. Suboptimality is defined as the average length ratio of all separate paths. According to the GPPC definition [41] high suboptimality is the result of optimality on long paths and highly suboptimal short paths. Since the length ratio depends mainly on long paths, a low ratio is also a sign of optimality on long paths and suboptimal short paths. Benchmarks with single maps from the GPPC shown in Section 3.5.3 demonstrate that our approach excels at more linear graphs like mazes and road networks (suboptimality of 1.0021 and length ratio of 1.00118 in maze-1550-15 with 2.25m nodes) while the suboptimality increases in maps with many large separated areas (maximum suboptimality of 1.0138 and length ratio of 1.0044 in room-800-80 with 0.625m nodes).

3.5.3 *Comparison to GPPC*

The Grid-Based Path Planning competition [41] provides a set of benchmarks [40] and hardware in order to compare grid based path finding algorithms. It consists of several grid based game maps and synthetic maps and has a total of 347,868 problems which are repeated five times resulting in 1.74 million searches. Since there is no current competition, and not all implementations and optimizations are accessible, we compare our approach with similar hardware and existing competition results (Intel E3-1220 DDR3 RAM (2011) vs. Intel E5620

DDR3 RAM (2010)). The relevant results from GPPC 2014 can be found in Table 3.1 including our approach with 25 and 100 fields.

Table 3.1: Timings, length ratio, suboptimality, storage and precomputation time of the GPPC 2014 benchmark compared to our results on similar hardware. † denotes parallel computation.

Algorithm	Total (s)	Length Ratio	Sub Opt	Storage	PreCmpt. (min)
RA*	492224	1.073	1.0580	0MB	0.0
JPS	108750	1.000	1.0000	0MB	0.0
BLJPS	25139	1.000	1.0000	20MB	0.2
BLJPS2	12947	1.000	1.0000	47MB	0.2
RASubgoal	2937	1.022	1.0651	264MB	0.2
NSubgoal	1345	1.000	1.0000	293MB	2.6
CH	630	1.000	1.0000	2.4GB	968.8
SRC-dfs-i	330	1.000	1.0000	28GB	†11650
Ours 100 fields	326	1.001	1.0043	11.8GB	15.0
Ours 25 fields	280	1.003	1.0110	3.2GB	3.7
PDH	256	1.078	1.1379	649MB	13.0
SRC-dfs	252	1.000	1.0000	28GB	†11650
Tree	50.9	1.135	2.1657	568MB	0.5

Table 3.2 displays the results for different numbers of distance fields. With an increasing number of fields, the search time increases due to the higher number of fields lookups. This is both due to the number of pre-computed fields itself and the more frequent field swaps. Note, that the GPPC benchmark contains a large amount of mazes and wide open rooms, which results in good suboptimality since our algorithm only requires a small amount of fields for these kinds of problems.

Table 3.2: Results of our approach for the GPPC benchmark with multiple numbers of fields on current hardware for 1.7 million search paths.

Fields	Total (s)	Length Ratio	Sub Opt	Storage	PreCmpt. (s)
4	819	1.0110	1.0409	0.80GB	19
5	464	1.0074	1.0314	0.91GB	24
10	262	1.0041	1.0197	1.48GB	45
25	239	1.0025	1.0110	3.20GB	111
50	232	1.0014	1.0066	6.05GB	218
100	280	1.0013	1.0043	11.76GB	433
200	328	1.0005	1.0024	23.19GB	871

3.5.4 *Comparison to A**

We use an optimized version of A* with a field containing flags instead of a closed list in order to test different settings of our approach and to validate its performance in other structures. For comparison, we pick random pairs of nodes and calculate the path length, skipping unconnected pairs. As benchmarks we use various structures as shown in Figure 3.7: road networks from the 9th DIMACS challenge [7], a mesh from the Stanford 3D Scanning Repository and several 2D Pathfinding Benchmarks [40]. Table 3.7 contains the results including timings, the percentage of suboptimal paths, the mean error of these paths, the total length ratio and the worst case ratio found during the benchmark. In the case of the Thai Statue, it is obvious that A* is much slower compared to the planar scenario. The reason for this is the fact that we use the euclidean distance as heuristic which is worse in 3D than in 2D. The same is true for the maze, where A* explores a large part of the graph in each iteration, again due to the inefficiency of the heuristic. While 69% of our paths are suboptimal in the case of the Thai Statue, their mean length error is only 1.11% and the total length ratio is below 1%. The reason for this high amount of suboptimal paths lies in the high branching of the mesh which makes the detection of artifacts difficult. The worst cases are all paths that are smaller than the width of the landmark areas and therefore have a significantly high upper bound compared to their actual length.

3.5.4.1 *Path quality improvements*

An effective approach to improving path quality is to take the center node of the calculated path as new node and to calculate the paths from the original start and target node to this center node. These can be joined and then compared to the original result, returning the shorter solution. Recursion can be achieved by splitting the sub-paths at their center nodes as well. This process can be repeated until the joined sub-paths are longer than the initial path or sub-path. Since the run time of our approach is linear in the path length, every split takes a similar amount of time as the original path calculation. Table 3.7 shows several results how this improves the quality of the solutions.

3.5.5 *Comparison to Contraction Hierarchies and Subgoal Graphs*

Contraction Hierarchies (CH) and Subgoal Graphs (SG) are the most promising algorithms from [41] with respect to both total time and preprocessing time. While SG is limited to grids, CH and its successors are widely used for navigation on street maps. In Table 3.3 we compare our approach to these methods using the GPPC benchmark set. We have split the data sets in categories in order to show in which cases any of the algorithms excel. This analysis shows that our approach and

CH are very slow for the maze-like maps, mainly for maze-1550-15, while SG has problems with the random maps.

Table 3.4 shows the comparison to CH on non-grid graphs, where it benefits from low branching while slowing down on graphs with higher branching like the Thai Statue mesh.

Table 3.3: Comparison of our approach to Contraction Hierarchies and Subgoal for the GPPC benchmark.

	Preprocessing in s			Pathfinding in s		
	Ours	CH	SG	Ours	CH	SG
Starcraft	33	5372	**31**	**7**	39	8
Dragon Age	15	456	**2**	7	13	**4**
Dragon Age 2	12	627	**2**	12	19	**7**
Maze	68	2894	**1**	230	190	**52**
Random	194	714	**23**	**11**	25	720
Room	112	20368	**7**	13	83	**10**
Total	434	30432	**66**	**280**	368	802

Table 3.4: Comparison of our approach to Contraction Hierarchies for multiple structures and 10000 random paths. More details for these graphs can be found in Table 3.7.

	Preprocessing in s		Pathfinding in s	
	Ours	CH	Ours	CH
New York	**4**	8	**0.5**	0.8
Western USA	128	**86**	5.2	**1.2**
Central USA	**403**	418	8.2	**4.6**
Thai Statue	**134**	81268	**1.8**	4.7

3.5.6 *Random placement of landmarks*

While the fixed placement of landmarks at largest distance to all others provides a well defined upper bound for errors and a uniform distribution with lower landmark to size ratios, it is sequential and the later landmarks happen to be placed in dead ends instead of highly frequented areas. Therefore random distribution simplifies parallel processing and can outperform the fixed placement at higher number of landmarks in both speed and overall quality, although there might be a minimal effect on the upper bound for errors. Table 3.8 shows the results of the fixed iterative placement in comparison with random placement. Most notable is that the quality of the maze-1550-15 map decreases for fixed placement from 10 to 25 fields while it increases steadily for the randomly placed landmarks. This is most likely due to distortions of the path caused by landmarks in corners and dead

ends. The GPPC benchmark benefits most from the random placement, while for the Thai Statue, the quality slightly decreases. This is a result of to the more uniform distribution over the mesh when using iterative placement and the fact that there are no dead ends on the mesh. It is also evident that the fifth field in the grid maps greatly increases the quality which also shows the importance of central landmarks.

3.5.7 *GPU implementation*

Table 3.6 shows the results of our GPU implementation in relation to the CPU equivalent. A portion of the time is consumed by the transfer of the full paths back to the host system, which is just an array of indices with fixed size. This could be further optimized by either persisting just the directions or by transferring only a partial path for online path finding. The graph including the connectivity and cost is simplified to a list of arrays. Due to high branching and collisions, for the larger graphs the GPU based pre-computation is not significantly faster than the CPU implementation, while the transfer time is negligible. The timings are shown in Table 3.5.

Table 3.5: Timings including transfer time for the GPU based distance field generation and speedup compared to the CPU counterpart.

	Radeon RX480		Radeon RX5700	
	Time(ms)	Speedup	Time(ms)	Speedup
New York	238	17	117	35
Western USA	38842	3	29716	4
Central USA	77012	5	59992	6
Thai Statue	22511	6	18763	7
GPPC	65059	7	15005	27

Table 3.6: Timings (including transfer time) for the GPU based pathfinding and speedup compared to the CPU variant. More details can be found in Table 3.7.

	Radeon RX480		Radeon RX5700	
	Time(ms)	Speedup	Time(ms)	Speedup
New York	47 (72)	7	29 (70)	7
Western USA	233 (266)	20	133 (184)	29
Central USA	329 (355)	23	251 (306)	27
Thai Statue	128 (158)	12	84 (136)	14
GPPC	8171 (10565)	27	5750 (9176)	31

Table 3.7: For multiple structures: timings, percentage of suboptimal paths, mean error of these paths and total length ratio. Recursion refers to splitting the path in subpaths and repeating the calculation as explained in Section 3.5.4.1.

DIMACS New York (264.346 nodes)
10000 random paths
time for A* : 71s, for field generation: 4.2s

	Time(s)	Suboptimal	Mean	Ratio	Worst
Normal	0.5	11%	7.21%	1.0083	3.36
Recursion 1	0.8	8%	5.79%	1.0052	2.97
Recursion 2	1.1	8%	4.86%	1.0040	2.41
Recursion 3	1.4	7%	4.56%	1.0036	2.41

DIMACS Western USA (6.262.104 nodes)
10000 random paths
time for A* : 2618s, for field generation: 128s

	Time(s)	Suboptimal	Mean	Ratio	Worst
Normal	5.3	12%	5.77%	1.0071	3.39
Recursion 1	9.6	8%	4.94%	1.0043	2.37
Recursion 2	12.8	7%	4.33%	1.0032	2.37
Recursion 3	16.1	7%	3.90%	1.0028	2.37

DIMACS Central USA (14.081.816 nodes)
10000 random paths
time for A* : 4944s, for field generation: 403s

	Time(s)	Suboptimal	Mean	Ratio	Worst
Normal	8.2	15%	3.41%	1.0054	2.31
Recursion 1	15.1	10%	3.10%	1.0033	1.96
Recursion 2	19.3	8%	2.60%	1.0023	1.58
Recursion 3	23.2	8%	2.34%	1.0020	1.58

Stanford Thai Statue (4.999.993 nodes)
10000 random paths
time for A* : 3383s, for field generation: 134s

	Time(s)	Suboptimal	Mean	Ratio	Worst
Normal	1.9	69%	1.11%	1.0077	1.74
Recursion 1	3.4	66%	0.93%	1.0062	1.47
Recursion 2	4.9	64%	0.87%	1.0056	1.45
Recursion 3	6.2	64%	0.85%	1.0055	1.45

GPPC random512-35-0 map (161.552 nodes)
10000 random paths
time for A* : 54s, for field generation: 2.3s

	Time(s)	Suboptimal	Mean	Ratio	Worst
Normal	0.4	27%	2.51%	1.0069	2.61
Recursion 1	0.6	21%	1.97%	1.0042	2.61
Recursion 2	0.9	16%	1.80%	1.0030	1.46
Recursion 3	1.2	15%	1.79%	1.0027	1.46

GPPC maze-1550-15 map (2.252.039 nodes)
10000 random paths
time for A* : 2015s, for field generation: 19s

	Time(s)	Suboptimal	Mean	Ratio	Worst
Normal	6.3	48%	0.21%	1.00118	1.37
Recursion 1	11.2	46%	0.17%	1.00111	1.17
Recursion 2	15.2	45%	0.14%	1.00105	1.06
Recursion 3	19.0	43%	0.12%	1.00099	1.05

3.6 DISCUSSION

As shown our approach is a serious competitor to current search algorithms when the computations are performed on the CPU. It is able to bridge the gap between optimal algorithms with high run time or high pre-computation time and suboptimal algorithms, while being easy to implement and very scalable with respect to memory consumption, run time and sub-optimality. The performance mainly depends on the memory latency and bandwidth, which are the bottlenecks in combination with branching and cache misses.

It is also usable for GPU path finding and profits from the fast graphics memory and bandwidth, though the memory might not suffice for the current high resolution textures coupled with many distance fields for large sized maps in games. The memory cost for all paths and the bus transfer time have to be taken into account. However, it is unlikely for current games to have this high amount of units being processed at the same second and requiring the complete path information.

While short paths are more prone to errors since the maximum error is bound by the distance to landmarks, a fallback to an A* algorithm could also be considered in this case.

3.6.1 *Path planning by minimal distance map*

In order to reduce the memory consumption, it is possible to fall back to the minimal distance map shown in Figure 3.3 for search. Through this map it is known which landmark is the closest to each node and the shortest path to it. In order to find a path from one node to another, it is only necessary to find a path between the two landmarks to which they belong. One solution is to store all paths between the landmarks, where the paths are given by the distance fields. So the distance fields are not more needed to be kept in storage, but the paths are.

An other solution is to generate a graph based on the minimal distance map with the landmarks as nodes. Edges between these nodes exist where a minimal distance field is neighbor to an other. The shortest paths and costs between the landmarks can be calculated using the minimal distance map. For this method it is not necessary to fully calculate the distance fields but just until there is no more change in the minimal distance map. Based on the graph a path can be searched, either by other algorithms or by an hierarchical approach.

Both approaches showed reasonable results but must be further looked into and optimized.

3.7 PROSPECTS

In a future work we want to look into suitable compression of the distance fields and measurements to classify problems in order to develop a heuristic to suggest or choose the most favorable settings. Furthermore, we plan to increase optimization of the GPU usage and reduce branching and cache misses. We also want to consider multi-agent scenarios with the combination of potential fields and different agent sizes. In connection with this we will evaluate potential fields in combination with our approach and how to handle local minima introduced by this method. A further step towards realistic path planning could be by introducing boundary conditions as suggested by Crane et al. [5]. Another interesting approach would be a mix of GPU and CPU computation, probably using OpenCL by computing an admissible graph on the GPU first and passing it for further processing to the CPU like in the split approach.

Table 3.8: Comparison of random selection of landmarks with selection by highest distance to all others. The results of the random selection are given in a 95% confidence interval. Better results are highlighted.

Fields	GPPC			
	Length Ratio		Suboptimality	
	Normal	Random	Normal	Random
4	1.0110	**1.0093-1.0106**	**1.0409**	1.0400-1.0451
5	1.0074	**1.0069-1.0077**	**1.0314**	1.0318-1.0345
10	1.0041	**1.0036-1.0038**	1.0197	**1.0188-1.0198**
25	1.0025	**1.0015-1.0018**	1.0110	**1.0105-1.0107**
50	1.0014	**1.0008-1.0009**	1.0066	**1.0069-1.0071**
100	1.0013	**1.0004-1.0004**	1.0043	**1.0044-1.0045**

Fields	DIMACS New York (264.346 nodes)			
	Length Ratio		Suboptimality	
	Normal	Random	Normal	Random
4	**1.1212**	1.0987-1.1750	**1.1583**	1.1462-1.2423
5	1.1004	**1.0769-1.1056**	1.1378	**1.1141-1.1490**
10	**1.0340**	1.0304-1.0461	**1.0621**	1.0538-1.0734
25	1.0131	**1.0095-1.0110**	1.0277	**1.0223-1.0247**
50	1.0052	**1.0039-1.0047**	1.0142	**1.0110-1.0129**
100	1.0025	**1.0015-1.0018**	1.0083	**1.0055-1.0063**

Fields	DIMACS Central USA (14.081.816 nodes)			
	Length Ratio		Suboptimality	
	Normal	Random	Normal	Random
4	1.1546	**1.0998-1.1622**	1.2169	**1.1506-1.2479**
5	1.1186	**1.0866-1.1226**	**1.1575**	1.1338-1.1855
10	**1.0348**	1.0350-1.0403	**1.0585**	1.0601-1.0688
25	**1.0107**	1.0105-1.0127	**1.0229**	1.0227-1.0263
50	**1.0040**	1.0041-1.0047	**1.0108**	1.0109-1.0124
100	1.0016	**1.0014-1.0016**	1.0054	**1.0049-1.0056**

Fields	Stanford Thai Statue (4.999.993 nodes)			
	Length Ratio		Suboptimality	
	Normal	Random	Normal	Random
4	1.1699	**1.1310-1.1524**	1.2499	**1.2001-1.2282**
5	**1.1194**	1.1121-1.1359	**1.1799**	1.1754-1.2019
10	**1.0430**	1.0546-1.0644	**1.0744**	1.0890-1.1043
25	**1.0150**	1.0197-1.0225	**1.0300**	1.0368-1.0411
50	**1.0063**	1.0083-1.0087	**1.0144**	1.0178-1.0186
100	**1.0031**	1.0037-1.0040	**1.0079**	1.0090-1.0096

Fields	GPPC random512-35-0 map (161.552 nodes)			
	Length Ratio		Suboptimality	
	Normal	Random	Normal	Random
4	1.1521	**1.1057-1.1411**	1.1778	**1.1448-1.1881**
5	**1.0918**	1.0875-1.1078	1.1202	**1.1169-1.1457**
10	**1.0409**	1.0415-1.045	1.0642	**1.0621-1.0671**
25	**1.0106**	1.0129-1.0157	**1.0196**	1.0240-1.0275
50	1.0057	**1.0050-1.0055**	**1.0110**	1.0107-1.0118
100	1.0034	**1.0022-1.0026**	1.0065	**1.0056-1.0062**

Fields	GPPC maze-1550-15 map (2.252.039 nodes)			
	Length Ratio		Suboptimality	
	Normal	Random	Normal	Random
4	1.0023	**1.0011-1.0019**	1.0034	**1.0025-1.0035**
5	**1.0008**	1.0012-1.0017	**1.0017**	1.0026-1.0033
10	**1.0005**	1.0010-1.0017	**1.0012**	1.0019-1.0027
25	1.0028	**1.0007-1.0012**	1.0031	**1.0016-1.0021**
50	1.0010	**1.0003-1.0007**	1.0020	**1.0011-1.0015**
100	1.0017	**1.0002-1.0004**	1.0023	**1.0008-1.0011**

4 CONCLUSION

It has been shown that it is possible to achieve scalable, realistic-looking solutions to hard problems, which can be used for games and simulations that are not vital, by reducing the problems to multiple exact solutions. In the case of the teeth, the computationally intensive subsurface scattering is broken down to a low resolution texture and scaled up, which is possible due to the scattering properties of the materials. And in the case of pathfinding, the number of landmarks and distance fields define the computational time and accuracy based on graph theory. The results are so close to the real solutions that it is difficult for the average observer to notice any anomalies.

4.1 PROSPECTS

Emerging artificial neural networks hold great promise for future developments in terms of realistic representations with minimal effort. However, they deliver approximations by design, are still very computationally intensive and only as good as their training data. While it is foreseeable that human teeth will soon be represented correctly and according to tooth color using these methods, however, not to the extent and in the form that would be necessary for dentists and the presentation of restorations. While there are neural network approaches to pathfinding, they also require pre-computation, i.e. learning, and are highly dependent on training data.

BIBLIOGRAPHY

[1] V. Blanz, T. Vetter, and A. Mehl. "A New Approach for Automatic Reconstruction of Occlusal Surfaces with the Biogeneric Tooth Model." In: *Int. J. Comp. Dent 8* (2005), pp. 13–25.

[2] Adi Botea, Martin Müller, and Jonathan Schaeffer. "Near optimal hierarchical path-finding." In: *Journal of Game Development* 1 (2004), pp. 7–28.

[3] Edwin Earl Catmull. *A subdivision algorithm for computer display of curved surfaces*. The University of Utah, 1974.

[4] Lenore Cowen and Christopher Wagner. "Compact roundtrip routing in directed networks." In: *Journal of Algorithms* 50 (2004).

[5] Keenan Crane, Clarisse Weischedel, and Max Wardetzky. "The Heat Method for Distance Computation." In: *Commun. ACM* 60.11 (2017), pp. 90–99.

[6] C. Dachsbacher and M. Stamminger. "Translucent Shadow Maps." In: *Proceedings of the 14th Eurographics Workshop on Rendering*. EGRW '03. 2003, pp. 197–201.

[7] Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson. "Implementation Challenge for Shortest Paths." In: *Encyclopedia of Algorithms*. Ed. by Ming-Yang Kao. 2008.

[8] Ariel Felner and Nathan Sturtevant. "Abstraction-Based Heuristics with True Distance Computations." In: *SARA 2009 - Proceedings, 8th Symposium on Abstraction, Reformulation and Approximation* (2009).

[9] A. Fernandez-Oliveras, M. Rubiño, and M. Perez. "Scattering anisotropy measurements in dental tissues and biomaterials." In: *Journal of the European Optical Society* 7.0 (2012). ISSN: 1990-2573.

[10] D. Fried, R. E. Glena, J. D. Featherstone, and W Seka. "Nature of light scattering in dental enamel and dentin at visible and near-infrared wavelengths." In: *Applied optics* 34.7 (Mar. 1995), pp. 1278–1285.

[11] J. R. Frisvad, T. Hachisuka, and T. K. Kjeldsen. "Directional dipole model for subsurface scattering." In: *ACM Transactions on Graphics* 34.1 (2014), 5:1–5:12.

[12] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. "Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks." In: 2008.

[13] Andrew Goldberg and Chris Harrelson. "Computing the shortest path: A* search meets graph theory." In: *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms* (2003).

[14] Kevin Grant and David Mould. *LPI: Approximating Shortest Paths using Landmarks*. Jan. 2008.

[15] X. Gu, S. J. Gortler, and H. Hoppe. "Geometry Images." In: *ACM Trans. Graph.* 21.3 (July 2002), pp. 355–361. ISSN: 0730-0301.

[16] Pat Hanrahan and Wolfgang Krueger. "Reflection from Layered Surfaces Due to Subsurface Scattering." In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '93. ACM, 1993, pp. 165–174.

[17] Daniel Damir Harabor and Alban Grastien. "Online Graph Pruning for Pathfinding On Grid Maps." In: *Proceedings of the AAAI Conference on Artificial Intelligence* (2011).

[18] P. E. Hart, N. J. Nilsson, and B. Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths." In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968).

[19] L. G. Henyey and J. L. Greenstein. "Diffuse radiation in the Galaxy." In: *Annales d'Astrophysique* 3 (Jan. 1940), p. 117.

[20] Satoru Horie and Alex Fukunaga. "Block-Parallel IDA* for GPUs (Extended Manuscript)." In: *CoRR* (2017).

[21] Henrik Wann Jensen, Justin Legakis, and Julie Dorsey. "Rendering of wet materials." In: *Rendering Techniques' 99*. Springer, 1999, pp. 273–281.

[22] Henrik Wann Jensen, Stephen R. Marschner, Marc Levoy, and Pat Hanrahan. "A Practical Model for Subsurface Light Transport." In: *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '01. New York, NY, USA: ACM, 2001, pp. 511–518. ISBN: 1-58113-374-X.

[23] Paul Kubelka and Franz Munk. "Ein Beitrag zur Optik der Farbanstriche." In: *Zeitschrift für technische Physik* 12 (1931), pp. 593–601.

[24] S. Kuo. *Color accuracy of digital images or use in craniofacial rehabilitytion*. 2003.

[25] C. T. Larsen. *Modelling and Rendering of Teeth*. 2011.

[26] C. T. Larsen, J. R. Frisvad, P. D. E. Jensen, and J. A. Bærentzen. "Real-Time Rendering of Teeth with No Preprocessing." In: *Advances in Visual Computing: 8th International Symposium, ISVC 2012*. 2012, pp. 334–345.

[27] M. I. A. Lourakis. *levmar: Levenberg-Marquardt nonlinear least squares algorithms in C/C++*. http://www.ics.forth.gr/~lourakis/levmar/. 2004.

[28] Ken Perlin. "Improving Noise." In: *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '02. ACM, 2002, pp. 681–682. ISBN: 1-58113-521-1.

[32] Szymon Rusinkiewicz and Marc Levoy. "Efficient variants of the ICP algorithm." In: *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*. IEEE. 2001, pp. 145–152.

[33] Christophe Schlick. "An Inexpensive BRDF Model for Physically-based Rendering." In: *Computer Graphics Forum* 13.3 (1994), pp. 233–246. ISSN: 1467-8659.

[34] Thomas W. Sederberg and Scott R. Parry. "Free-form Deformation of Solid Geometric Models." In: *SIGGRAPH Comput. Graph.* 20.4 (Aug. 1986), pp. 151–160. ISSN: 0097-8930.

[35] S. Shetty. *Layered Rendering Model for Human Teeth*. 2010.

[36] S. Shetty and M. Bailey. "A physical rendering model for human teeth." In: *International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2010, Poster Proceedings*. 2010.

[37] Jeremy Shopf, Joshua Barczak, Christopher Oat, and Natalya Tatarchuk. "March of the Froblins: Simulation and Rendering Massive Crowds of Intelligent and Detailed Creatures on GPU." In: *ACM SIGGRAPH 2008 Games*. SIGGRAPH '08. 2008.

[38] D. Spitzer and J. J. Ten Bosch. "The absorption and scattering of light in bovine and human dental enamel." In: *Calcified Tissue Research* 17.2 (1975), pp. 129–137. ISSN: 1432-0827.

[39] Wolfgang Straßer. "Schnelle Kurven- und Flächendarstellung auf grafischen Sichtgeräten." In: 1974.

[40] N. Sturtevant. "Benchmarks for Grid-Based Pathfinding." In: *Transactions on Computational Intelligence and AI in Games* 4.2 (2012).

[41] Nathan R. Sturtevant, Jason Traish, James Tulip, Tansel Uras, Sven Koenig, Ben Strasser, Adi Botea, Daniel Harabor, and Steve Rabin. "The Grid-Based Path Planning competition: 2014 entries and results." In: *Proceedings of the Eighth Annual Symposium on Combinatorial Search*. Association for the Advancement of Artificial Intelligence (AAAI), 2015.

[42] T. Uras, S. Koenig, and C. Hernández. "Subgoal graphs for optimal pathfinding in eight-neighbor grids." In: *Proceedings of the Twenty-Third International Conference on International Conference on Automated Planning and Scheduling*. ICAPS'13. 2013.

[43] Wikipedia. *Dental restoration*. 2019 (accessed March 25, 2019). URL: https://en.wikipedia.org/wiki/Dental_restoration.

[44] Lance Williams. "Casting Curved Shadows on Curved Surfaces." In: *Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '78. New York, NY, USA: Association for Computing Machinery, 1978, pp. 270–274. ISBN: 9781450379083.

[45] Lance Williams. "Casting Curved Shadows on Curved Surfaces." In: *SIGGRAPH Comput. Graph.* 12.3 (Aug. 1978), pp. 270–274. ISSN: 0097-8930.

[46] Chenglei Wu, Derek Bradley, Pablo Garrido, Michael Zollhöfer, Christian Theobalt, Markus Gross, and Thabo Beeler. "Model-based Teeth Reconstruction." In: *ACM Trans. Graph.* 35.6 (Nov. 2016), 220:1–220:13. ISSN: 0730-0301.

[47] Yichao Zhou and Jianyang Zeng. "Massively Parallel A* Search on a GPU." In: *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. Ed. by Blai Bonet and Sven Koenig. 2015.

[48] J. R. Zijp, J. J. Ten Bosch, and R. A. J. Groenhuis. "HeNe-laser light scattering by human dental enamel." In: *Journal of dental research* 74.12 (1995), pp. 1891–1898.

[49] J.R. Zijp. *Optical properties of dental hard tissues*. Doctoral dissertation. 2001.

[29] Maximilian Reischl, Evgenij Derzapf, and Michael Guthe. "Physically Based Real-Time Rendering of Teeth and Partial Restorations." In: *Computer Graphics Forum* 39.1 (2020), pp. 106–116. ISSN: 1467-8659.

[30] Maximilian Reischl, Christian Knauer, and Michael Guthe. "Using Landmarks for Near-Optimal Pathfinding on the CPU and GPU." In: *Pacific Graphics Short Papers, Posters, and Work-in-Progress Papers*. The Eurographics Association, 2020. ISBN: 978-3-03868-120-5.

[31] Maximilian Reischl, Christian Knauer, and Michael Guthe. "Parallel Near-Optimal Pathfinding Based on Landmarks." In: *Comput. Graph.* 102.C (Feb. 2022), pp. 1–8. ISSN: 0097-8493.