Karlsruhe Institute of Technology

Irradiance Importance Sampling

Bachelorarbeit
von

Alisa Jung

An der Fakultät für Informatik
Institut für Visualisierung and Datenanalyse (IVD)

Erstgutachter:	Prof. Dr. Carsten Dachsbacher
Zweitgutachter:	Prof. Dr. Hartmut Prautzsch
Betreuender Mitarbeiter:	Dipl-Inf. Florian Simon

Bearbeitungszeit: 02.10.2014 – 02.02.2015

Abstract

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Zusammenfassung

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Inhaltsverzeichnis

1. Introduction

[Kaj86]

2. Basics

2.1 Rendering Equation

2.2 Monte Carlo Integration

Monte Carlo Integration is a non-deterministic numeric method for computing integrals. We will start by introducing some basic prinicples of probability theory before explaining Monte Carlo Integration itself.

Monte Carlo Integration eignet sich gut, um hochdimensionale Integrale numerisch zu lösen. Wir werden dieses Verfahren später zur Beleuchtungsberechnung einsetzen.

- Wahrscheinlichkeitstheorie: Grundlagen
  - Was ist eine pdf
  - Was ist Erwartungswert und Varianz
- Monte Carlo Integration: Umformung mit durch pdf teilen
- Vorteile (siehe Veach Seite 39 (bzw pdf seite 65)
  - Konvergiert mit $\mathcal{O}(N^{-0.5})$ $\Rightarrow$ gut für hochdimensionale Probleme wie hier
  - simple: only sampling and point evaluation
  - allgemein
  - funktioniert bei Singularitäten
- Importance Sampling

2.3 Multiple Importance Sampling

Importance sampling als variance reduction method

Multiple Importance Sampling: Wie kombiniert man am Besten?

3. Path Tracing

3.1 Rendering Equation Path Space

We will now introduce an alternative formulation of the rendering equation measured over path space. This formulation can be easily converted to a basic path tracing algotihm, which we can modify and expand later on.

(TODO Gleichung einführen)

Using equation (naja die Gleichung halt) we have transformed the light transport problem to an integration problem. However, since the number of dimensions (i.e. number of possible path lengths) is theoretically infinite, we will solve this integral numerically using Monte Carlo Integration. (TODO)

Our final goal is to compute the incoming radiance at any pixel j:

(blablabla)

Form für Monte Carlo:

$$I_j = \int_\Omega f_i(X) d\Omega(X) \tag{3.1}$$

with $X = (x_0, \ldots, x_k)$ being a light transport path of an arbitrary length k and

$$f_i(X) = W_i(x_1 \to x_0) \cdot G(x_0 \leftrightarrow x_1) \cdot L_e(x_k \to x_{k-1}) \cdot \prod_{j=1}^{k-1} (f_r(x_{j+1} \to x_j - > x_{j-1}) \cdot G(x_j \leftrightarrow x_{j+1})) \tag{3.2}$$

and

$$d\Omega(x_0 \ldots x_k) = dA(x_0) \times dA(x_1) \times \cdots \times dA(x_k) = d\Omega(X^k) \tag{3.3}$$

We are integrating over surfaces since the paths in our path space are sequences of surface points.

Finally, we use Monte Carlo Integration to estimate the solution of equation 3.1:

$$\int_\Omega f_i(X) d\Omega(X) = E(\frac{f_i(X)}{p(X)}) \approx \frac{1}{N} \sum_{j=1}^{N} \frac{f_i(X_j)}{p(X_j)} \tag{3.4}$$

where $p(X)$ is a probability density function for sampling a certain path. Remember that $p(X)$ has to be > 0 wherever $f_i(X) \neq 0$. The next part will cover how to find a good pdf.

3.2 Importance Sampling Options

Rendering Equation (grob): $L_o(x, w_o) = \int f(x, w_o, w_i) \cdot L_i(x, w_i) cos(w_i) d\sigma(w_i)$

$\Rightarrow$ Wert im Integral wird potentiell "'groß"' für

- $cos(w_i)$ groß $\rightarrow$ mehr senkrecht weg sampeln
- $f(x, w_o, w_i)$ groß $\rightarrow$ sample bsdf
- $L_i(x, w_i)$ groß =>
  - next event estimation (explicit paths) (?)
  - cache and sample approximation of irradiance

3.2.1 cosine to normal

Pro: Am einfachsten. Con: Nicht so toll.

3.2.2 BSDF

Für was ist das eher gut geeignet, für welche Fälle eher nicht (siehe veachszene mit 4 verschieden großen Kreisen)

3.2.3 Next Event Estimation (sample light source explicitly)

Für was ist das eher gut geeignet, für welche Fälle eher nicht (siehe veachszene mit 4 verschieden großen Kreisen)

3.2.4 Irradiance caches

Taddaa, darum gehts dann nachher.

3.3 Path contribution

Wie sieht dann genau der Beitrag von einem Pfad speziell aus? (dazu fällt mir jetzt spontan nicht so viel zu schreiben ein) Falls das nicht oben schon steht hier mal noch die Measurement Contribution Function mit Skizze, vllt Pseudocode von dem Pathtracer und einmal nachgerechnet, dass tatsächlich das rauskommt oder so.

3.3.1 Probability

Woher kommt die Wahrscheinlichkeit? Naja, einfach für jedes weitere Sample die Wahrscheinlichkeit aufmultiplizieren. Vielleicht erklären, wie man überhaupt pdfs für Bsdf, Cos, ... kriegt. Vllt das auch weiter oben schon machen. Die pdfs für die Envmaps kommen eh später nochmal genauer.

3.3.2 MIS Weights

Die Möglichkeiten MIS Gewichte zu holen stehen schon weiter oben, wo MIS vorgestellt wird. Hier nur das, was ich am Ende wirklich verwendet habe. Bzw falls ich mehrere Sachen ausprobiere, halt alle die Sachen.

Was im Code grade auch noch fehlt, ist Sampling nach der BSDF und nach den Envmaps zu kombinieren. Zumindest bei der veach Szene mit den vier verschieden großen Lichtern und verschieden glänzenden Flecken hat das Sampling nach den Envmaps die selben Probleme wie die expliziten Pfade.

3.4 Russian Roulette

4. Irradiance Caching

The main goal of this thesis is to approximate the irradiance across a scene using local caches. This chapter will cover all relevant preprocessing steps: Chosing a cache representation, placing the caches in the scene, and filling them. The next chapter will cover the actual rendering process and describe how these caches can be used in a path tracer to estimate irradiance.

4.1 Environment Map Parameterization

Every cache for incoming radiance at a certain surface point is represented as a small local environment map. Every texel of this environment map will later represent the relative incoming radiance around the cache's position from the direction covered by that texel.

We chose the Octahedron representation proposed by [ED08]. The main advantange of this parameterization is that in comparison to cube maps and sphere maps, octahedron environment maps show the smallest variation of the solid angle covered by one texel. Since all probabilities in our path tracing algorithm are measured over solid angles, this choice results in the least distortion when converting probabilities from texel to solid angles, as shown in chapter (todo später) Additionally, the under- and oversampling in worst-case areas is still better than the worst over- or undersampling occuring in other known parameterization. (TODO der Satz ist doof. Ich will sagen, dass bei Oktaedern am ehesten noch gleichmäßig gesampelt wird)

To map a direction $d = (d_x, d_y, d_z)$ on a texel coordinate $p = (p_x, p_y)$, we first use planar projection to convert the direction vector to a point on the unit octahedron surface:

$$d' = \frac{d}{|d_x| + |d_y| + |d_z|} \tag{4.1}$$

Next we need to project this point on a two-dimensionall texture coordinate p. There are several ways to do this, we chose one that yields a rectangular texture:

$$p = \begin{cases} (d'_x - d'_z - 1, d'_x + d'_z) & d'_y \geq 0 \\ (d'_z - d'_x + 1, d'_x + d'_z) & d'_y < 0 \end{cases} \tag{4.2}$$

The resulting coordinate is in $[-2, 2] \times [-1, 1]$, with directions of the upper hemisphere being mapped to $[-2, 0] \times [-1, 1]$ and directions of the lower hemisphere being mapped to

$[0, 2] \times [-1, 1]$. As seen in figure (TODO), texels close to $p_x = 0$ belong to similar directions on adjoint octahedron surfaces.

This projection is relatively easy to compute forth and back, plus the resulting texels cover the whole area of the texture. Since both hemispheres are mapped to separate parts of the texture we can simly ignore the lower hemisphere at non-transmitting surfaces, when irradiance from below has no contribution to the outgoing light. We can then only use irradiance arriving at the upper hemisphere by only using - and storing - the left half of the environment map, which also saves a considerable amount of storage space.

4.2 Creating Caches

Actually creating these caches consists of 4 steps: Photon Mapping, distributing the positions of the caches over the scene, actually filling the caches and storing them. Since we intend to fill the caches with incoming radiance, we need some way to approximate the irradiance first. We do this with a modified version of photon mapping. When the photon mapping is done, the photons are used to place and fill the caches.

4.2.1 Modified Photon Mapping

Photon mapping itself was first introduced by (TODO short introduction of photon mapping). The original idea is to trace a huge number of emitted photons from all light sources along their paths and store a photon at every surface point along that path. (TODO was ist photonmapping)

A photon path is created by sampling a point and outgoing direction for a photon and then tracking it accross the scene. Whenever there are multiple light sources, a light source L is chosen with a probability p_L proportional to its surface:

$$p_L = \frac{Area_L}{\sum_{alllightsources L'} Area_{L'}} \tag{4.3}$$

For simplicity's sake we're only considering area light sources here. A point x on light source L is sampled uniformly with probability $p_x = 1/Area_L$, the probability for sampling direction ω_o at this point is p_ω (TODO was ist mit der Richtung?). Given a light source L, let $p = p_\omega \cdot p_x$ be the probability of sampling point x and outgoing direction ω_o on that light source and $\Phi_{L,e}(x, \omega_o)$ the power of that light source at the given point and direction. When N photons are emitted, the initial energy of the photon emitted from light source L at x into ω_o is given as

$$\frac{\Phi_{L,e}(x, \omega_o)}{p_L \cdot p \cdot N} = \frac{\Phi_{L,e}(x, \omega_o) \cdot \sum_{alllightsources L'} Area_{L'}}{p_d \cdot N} \tag{4.4}$$

This photon is then traced accross the scene. Whenever the photon's path intersects a diffuse surface, a photon with the current energy is stored. At any intersection y, no matter if it's diffuse or specular, the outgoing direction ω_o of the photon path is sampled from the surface's bsdf f and the incoming direction ω_i. (TODO hier adjoint? bzw. hier wird Importance transportiert, das halt.). The path is then continued into the sampled direction and the energy is weighed with the bsdf's value $f(\omega_i, y, \omega_o)$ divided by the probability of sampling ω_o.

Since we plan to use photon mapping to approximate irradiance instead of rendering an image we need to make some adjustments to this process. One of them are the stored

photons themselves: Each of our photons consists of its position including the local surface coordinate system, its incoming direction, a pointer to the surface's bsdf and its energy. The next section will explain why the local coordinate system is necessary. The bsdf is needed in case we want to place a cache at the photon's position: It tells us whether we need an environment map for both hemispheres or only one. We don't need the actual energy distribution over different wavelengths; storing a photon's average energy in a single float value is sufficient.

Besides, we only need the relative irradiance from all directions to create a cache. This allows us to ignore the constant factors N and (summe über lichtflächen) when computing a photon's initial energy and use

$$\frac{\Phi_{L,e}(x, \omega_o)}{p_\omega} \tag{4.5}$$

instead. To avoid unnecessary computing, we cancel photon paths when their accumulated energy falls too close to zero, when the path length reaches 32 and using russian roulette. When all photon paths are done, all created photons are stored in a kD-Tree.

4.2.2 Placing Caches

There are two interesting areas for placing caches: Those that have a lot of action going on lightwise, and those that are potentially hit very often when using path tracing later on.

To cover the latter area, we will shoot camera rays through every few pixels. If the bsdf at their first intersection with the scene is smooth, we place a cache at that position. Otherwise the ray will be traced until it leaves the scene or until a smooth surface is hit and we can place a cache.

The other cache positions are created from the first n photon positions created by photon mapping. This will create many caches at areas that have many photons arriving at them, and place less caches in areas where only few photons intersect the scene. Since the photon paths are created randomly, (TODO a similar?) the distribution of cache positions generated from simply taking the first n photons will be similar (on average) to choosing the photons randomly (TODO hoff ich doch...).

Note that from now on we have no information concerning the surface properties (bsdf, geometry, normals) at the cache position, only the positions themselves. To avoid inconsistent coordinates later on, it is necessary to give each cache its local coordinate system. Luckily we already stored these systems with every photon and can simply use these for the photon caches. The camera caches get their coordinate system from the intersection information which is still available when the caches are placed in the scene.

At this point we also need to decide whether to create a cache for irradiance from both hemispheres or only one. To do so, we look at the bsdf (which is either stored with the photon or can be retrieved from the intersection information for camera caches): If it is transmitting, we create an empty cache for both hemispheres and store it at the position with its coordinate system. Otherwise a cache for the upper hemisphere is created and stored. Practically the only difference between these two caches is the length of the array containing the environment map's texels. We don't consider materials where only irradiance from below contributes to radiance leaving at the upper side.

4.2.3 Filling a Cache

After the caches are created and placed across the scene we need to fill them with an approximation of the irradiance around them. The number k of photons used to fill a cache is fix and can be adjusted before starting to render the image. Filling a cache works like this:

$photonlist \leftarrow kdTreePhotons.getKClosestPhotons(k, cache.position)$
for all *Photon* p in $photonlist$ **do**
  $incoming\_direction\_local \leftarrow cache.transformToLocal(p.direction)$
  if $cos(local\_incoming\_direction) > 0 \;||\; cache\_for\_both\_hemispheres$ **then**
    $texel \leftarrow cache.convertDirectionToTexel(incoming\_direction\_local)$
    $cache.environmentMap[texel] += p.energy$
  end if
end for

Note that the actual number of photons contributing to the cache may be lower than k:

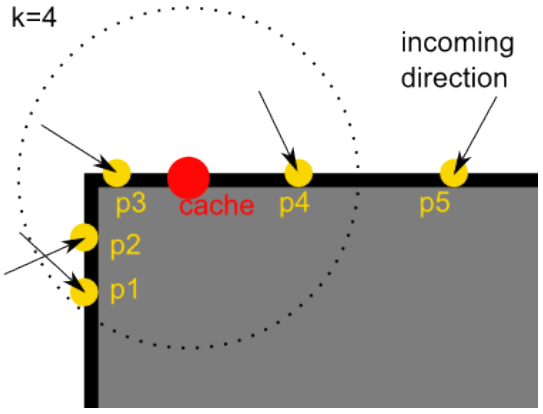

Abbildung 4.1: When filling the cache with 4 photons, photons p1 to p4 are considered and p5 is ignored. However, the incoming direction of p2 results in a negative cosine with the cache's surface normal, meaning that the photon arrives from below from the cache's point of view. Thus p2 is discarded for this cache and only p1, p3 and p4 are used to fill it.

Whenever a photon is used to fill a cache, its incoming direction is converted into a texture coordinate for our octahedron environment map (see equations 4.1 and 4.2). The energy of the photon is then added to that texel. (TODO bilinear filtern). It is also possible to weigh the energy contribution of a photon with its distance to the cache's position, but we didn't notice any differences in the resulting images compared to the procedure above.

4.2.3.1 Create a useful probability density function

The next step is to convert the filled environment map to a probability density function that can be used to sample an outgoing direction proportional to the irradiance during path tracing. The main idea is to normalize the sum of all texels of the environment map to 1 and then sample directions according to the probability values stored in the texels. Obviously, the first step towards a useful pdf is simply that: add up all energy values from the environment map and divide every texel's value by that sum.

However, there's a catch: Monte Carlo Integration requires the pdf to be greater than zero wherever the integrated function contributes to the integral. So we have to make sure that our pdf isn't equal to 0 when irradiance arriving from the corresponding direction can

contribute to the integral. With smooth bsdfs that is the case for all directions (in the upper hemisphere), so we have to set all texels with value 0 to a bigger value. This value should be big enough so it won't cause any floating point precision problems - either by being ignored completely in the final pdf or by causing fireflies due to dividing by very small numbers. On the other hand the value should not affect the quality of the irradiance approximation stored in non-zero texels.

We tried several options: The first one was setting the zeros to the smallest positive value from all other texels before normalizing. However, in many cases this value turned out to be too close to zero and disappeared completely after the normalization. Even normalizing, setting all 0 texels to the smallest value after normalizing, and normalizing again resulted in near-zero values that didn't work well in the final image.
Next we tried setting every value below half average of all non-zero values to half average. The reason for that idea was that many caches had some texels with values around some 100, but a lot more texels containing positive values below 1. This approach caused by far the most noise in the final image.
After looking at several normalized environment maps, we tried to simply set all values smaller than 0.0002 to 0.0002 and normalize again. This seemed to be working quite well so far. We even tried combining this approach with the first one, but as it turned out the minimal value was always smaller than 0.0002, regardless of whether it was picked before or after the first normalization. Different thresholds from 0.0001 to 0.001 didn't produce any noticable differences in the final images.

There's also a reason not to set every 0 to a fix non-zero value before the first normalization: The sums of two caches can easily differ in an order of 2 magnitudes. So while setting all 0 texels to, say, 1 may work perfectly fine for one cache with a (previous) sum of some 100, it can totally destroy the irradiance approximation for another one with a sum of 50 or just disappear again for caches in bright regions with sums up to some 1000. Additionally, the initial sum of a cache depends on the number of photons used to fill it and the overall energy of these photons, while the average value of a texel also depends on the overall resolution of the octahedron map. Considering all these uncertainties it seems more reasonable to increase the 0-texels after a first normalization, when we can make a more precise statement about the overall properties of the cache.

The values of the environment map now form a valid probability density function that can be used to sample directions. To improve computation time we also create a second environment map containing the cumulative density function for that pdf and added it to the cache. See chapter (wo gerendert wird) on how samples are torn from these environment maps.

4.2.3.2 Convert probabilities to solid angle

Generating samples is not the only purpose of these caches. The other one is to get the probability linked to the sample, and to compute the probability of sampling a given direction from a cache. These probabilities are needed to compute weights for multiple importance sampling and to compute the probability of sampling a path. With basic path tracing probabilities are usually measured over solid angle, while our pdf is measured over a rectangle (or square in case only the upper hemisphere is important). Its width and height correspond to the number of rows and columns of the environment map, and each texel covers the area of one unit square. Remember that the pdf values were normalized to sum up to one.

Unfortunately, there is no easy way to analytically compute the solid angle covered by each unit square in a rectangle. Ignoring this problem and directly using the probability value from a cache resulted in images like figure (TODO Bild wo überhaupt nicht korrigiert

wird). Approximating the solid angle measure by multiplying with $\frac{2\pi}{size\_of\_environmentmap}$ (TODO macht es hier noch Sinn, durch die Größe zu teilen?) isn't enough either: The overall energy of the resulting images seemed about right, but it was rather weirdly distributed (TODO Bild). So we created another map to approximate the solid angle covered by each texel:

```
solid_angle_map ←new octahedronMap                    ▷ upper hemisphere only
numSamples ← 1000 · solid_angle_map.length
for i = 0 to numSamples do
    Vector3 sample ← createRandomSampleOnHemisphere()
    Vector2 texel ←solid_angle_map.convertDirectionToTextureCoordinate(sample)
    index ← ⌊texel.x⌋ + solid_angle_map.width*⌊texel.y⌋
    solid_angle_map[index] ++
end for
for (i = 0 to solid_angle_map.length) do
    solid_angle_map[i] / = numSamples                 ▷ normalize to 1
    solid_angle_map[i] * = 2π                         ▷ the integral over one hemisphere is 2π
end for
```

At the end every texel represents the relative solid angle covered by its area. Central texels and those close to the border represent directions close to the surface's normal and along the surface itself respectively and will hold smaller values. The areas in the middle of each octahedron surface are close to parallel to the unit hemisphere and thus cover a bigger solid angle.

A pdf value from a texel in a cache's environment map can now be converted to a probability value measured over solid angle by dividing it by the value from the corresponding texel in the solid angle map. This can be done for all texels in all environment maps in a preprocessing step, as the number of accesses to caches during the actual rendering may easily exceed the total number of texels.
The solid angle map can also be doubled and then applied to the lower hemisphere. In that case its values will sum up to 4π, which equals the integral over the unit sphere.

4.2.4 Storing Cache

All caches are stored in a kd-Tree for fast and easy access. The kd-Tree containing the photons won't be needed again and can be deleted. Each cache now consists of the following components:

- An octahedron environment map containing the probability density function, measured over solid angle, that approximates the irradiance around the cache. We'll call this one *pdf_map*
- An additional map containing the cumulative density function for faster sampling, called *cdf_map*. Note that the cdf is created after the last normalization, but before the pdf is adapted to the solid angle measure.
- Its position
- Its local coordinate system

5. Rendering

This chapter will describe how the irradiance caches from the previous chapter can be used to improve path tracing. The first part covers how directions can be sampled from the environment maps and how this relates to the rendering equation / measurement contribution function / whatever. The second part combines the sampling of envmaps with sampling of the BSDF and next event estimation using multiple importance sampling. Hopefully, using irradiance caches will make next event estimation unnecessary, but if both methods shall be used, there are equations to consider.

We will use the terms "'bsdf sampling"' and "'irradiance sampling"' for the process of sampling an outgoing direction and its probability from a surface point's bsdf or corresponding irradiance cache respectively. Taking samples from a bsdf will not be discussed here any further.
The first part of this chapter will cover generating samples and probabilities from an irradiance cache. After that we describe how we combined bsdf sampling, irradiance sampling and next event estimation using multiple importance sampling and evaluate the results.

5.1 Sampling Caches

5.1.1 Extend Path

As described in chapter (TODO), path tracing basically creates random paths starting at the camera and traces these paths across the scene until either a light source is hit or the path leaves the geometry. To do that, we need to choose a new direction every time the path hits a surface at an intersection point x. This is normally done by sampling an outgoing direction from the surface's bsdf. In that case the applied pdf is proportional to the bsdf term from the rendering equation. The goal of irradiance caching is to use a pdf which approximates the irradiance $L_i(\omega_i, x)$ instead of the bsdf.

Our basic algorithm for sampling a new outgoing direction for a path proportional to the irradiance consists of several steps: First we have to choose the cache we want to use. Sometimes, especially at corners, thin walls or curved surfaces, the closest cache can have another normal than the surface at x itself. If the deviation is small enough, the cache can be used anyway. However, if the cache's rotation is too severe, it might be better to pick one of the next closest caches more parallel to the surface at x. We chose 0.9 to be the threshold for the cosine between the cache's and surface's normal and never considered more than the 5 closest caches.

This may seem arbitrary, but in our example scenes most caches violating that threshold had a cosine of 0 or less to the surface. Plus whenever a cosine was positive but below 0.9, some of the other closest caches often exhibited bigger values. We also didn't experience any problems with spheres, since the caches were always placed densely enough.

With more complex scenes and small spheres in rarely lit areas that are not seen by the camera directly but hit many times for indirect lightning, one would have to test if 0.9 is still a good choice or if a smaller values would be a better option. On the other hand, if not many photons arrive in a region in the first place, it might prove more useful to just use normal bsdf sampling there.

Instead of always choosing the closest cache (or that one of the 5 closest ones with the best cosine), we decided to randomly pick one of those caches that was among the 5 closest to x and also had the best cosine value. In most cases these were all 5 closest caches with a cosine of 1.

Algorithm 1 Pick Cache

```
1:  procedure PICKCACHE(surfacePoint)
2:      cacheList ← kdTreeCaches.getKClosestCaches(5, surfacePoint.position)
3:      bestCos ← −1
4:      for i = 0 to 4 do
5:          currentCache ← cacheList[i]
6:          currentCos ← cos(currentCache.normal, surfacePoint.normal)
7:          if currentCos > bestCos then      ▷ only allow caches with best cosinus value
8:              bestCos ← currentCos
9:              bestCache ← currentCache
10:         else if currentCos == bestCos ∧ getRandomNumber() > 0.6 then
11:                     ▷ Choose randomly between those caches with shared best cosine
12:             bestCache ← currentCache
13:         else                                                  ▷ Ignore this cache
14:         end if
15:     end for
16:     if bestCos > 0.9 then
17:         return bestCache
18:     end if
19:     return null                                          ▷ sample bsdf instead
20: end procedure
```

As soon as the cache is determined, the next step is actually sampling a direction from that cache. For that purpose three random numbers between 0 and 1 are required. The first one is used to pick a texel from the cache's environment map. To do that we look for the field in the cache's cdf array (see chapter TODO) with a value bigger or equal to the random number. The other two are needed to sample a point whithin the texel that can be converted to a direction (see equations TODO umrechnung octahedron map).

As one also needs the probability of the sampled direction whenever a direction is sampled, this function also returns the probability for sampling the direction. Note that while the cdf was created from a pdf that sums up to one, the returned probability is measured over solid angle and was computed after the cdf was created.

Algorithm 2 Sample Direction from Cache

1: ▷ pdfValue will be filled with probability
2: **procedure** SAMPLEDIRECTION($cache, pdfValue, rand1, rand2, rand3$)
3: ▷ use binary search to find index with first value ≥ rand1
4: $texel\_index \leftarrow$ BINARYSEARCH(cache.cdf,rand1)
5: ▷ extract probability over solid angle from pdf map
6: $pdfValue \leftarrow cache.pdf\_map[texel\_index]$
7: ▷ index to 2D texture coordinates in $[0, mapWidth] \times [0, mapHeight]$
8: $texel.x \leftarrow texel\_index \% cache.mapWidth + rand2$
9: $texel.y \leftarrow texel\_index / cache.mapWidth + rand3$
10: ▷ transform to (local) direction
11: $Vector3D\ direction \leftarrow cache.texelToDirection(texel)$
12: ▷ transform to world coordinates
13: $direction \leftarrow cache.coordinateSystem.toWorld(direction)$
14: **return** *direction*
15: **end procedure**

After a (global) direction ω is sampled, we can continue the path along ω. The last thing left to do is to weight the energy carried along the path. Therefor we have to evaluate the bsdf at x with ω as incoming (!) direction ω_i and use the direction where the path came from as outgoing direction ω_o. We then can weight the energy according to equation (TODO):

Algorithm 3 Weight path energy for Monte Carlo Integration

$bsdfValue \leftarrow x.bsdf.eval(\omega_i, x, \omega_o)$
$cos \leftarrow cos(\omega_i, x.normal)$
$pathEnergy \leftarrow pathEnergy \cdot bsdfValue \cdot cos / pdfValue$

Note that the bsdf technically evaluates directions represented in local coordinates at x, so we would actually have to convert the used directions from world to local coordinates first. Fortunately, if the cosine of the used cache happens to be 1, we can skip the transformation of ω_i and just use the local cache coordinates from line 11 instead, as long as we can guarantee consistency of both coordinate systems.

However, there are cases when sampling for irradiance is not a good option: At specular surfaces, only light from one direction (or two for transmitting materials) actually contributes to the integral to compute. It is therefore unnecessary to randomly sample several directions, we can directly sample the one (or 2) important direction from the surface's bsdf. Note that in chapter (TODO) we only placed caches at non-specular surfaces: The cache positions where either created from photons (which are only stored when the bsdf was smooth) or from camera rays (which were traced across the scene until a smooth surface was found).
It might also happen that there are no caches nearby or the closest caches are too much rotated to provide useful information. We then resort to regular bsdf sampling.

5.1.2 Compute probabilities for given directions

For simple path tracing with implicit paths only and irradiance sampling when possible the previous section contains all steps we need. However, if we want to combine irradiance sampling with next event estimation or bsdf sampling, we also need to be able to extract a probability value from a cache for a given direction.
Therefor we first pick a cache just like in algorithm 1. Next we convert the direction to local coordinates in the cache's coordinate system and then on to texture coordinates according

to equation (TODO). A simple acess to the cache's *pdf_map* at the computed texel yields the probability for having sampled the given direction with this cache, measured over solid angle.

5.2 Combining BSDFs and Caches

Figures (TODO) show the flaws of only chosing one local sampling method. As opposed to combining bsdf sampling and next event estimation, bsdf and irradiance sampling can not be done at the same time: Since we use path tracing, we don't want to split up a path into two paths whenever we hit a surface. Hence we need a way to choose between bsdf and irradiance sampling and a way to combine them.

We use the bsdf roughness to decide whether to sample the bsdf or irradiance. If the bsdf is ideally diffuse, the bsdf itself will have the same value for every direction and we can't gain much information or increase the integrand by sampling the bsdf. On the other hand, if we hit a (near) delta bsdf, for most of the directions the irradiance will have no contribution due to the bsdf being close to zero for most angles.

There are certain reasons not to sample the irradiance at all. One are delta-bsdfs: Whenever a surface is perfect specular, perfect transmitting or both, there is only one (or two) directions that actually contribute light to the current path. Whenever we hit a surface with a delta-bsdf, we will automatically only sample the bsdf with MIS weight 1, and also skip the next event estimation completely, since the probability of it having any contribution at all is zero.

Another case depends on the placement of the caches. Sometimes there might just be no caches available close to the current surface point. In simple scenes, this is almost never the case, but we can never be sure to prevent this completely, especially with more complex scene.
A similar case is a more complex, curved geometry: Consider placing caches on a sphere. Since we can never cover the whole surface area of a sphere with points, there will always be points that are hit by camera rays that have no cache at their exact position. There might be a cache very close to them, but judging from an analytical point of view the angle between their normals will never be 0. If that angle is still small (or the cosine between them big) enough, we can ignore the deviation and still use the cache, for everything we have was only approximated in the first place. We allowed a small margin of difference between the cache's normal and the surface point's normal. But whenever the cosine is smaller than 0.9 (TODO - dabei lassen?) we ignore the caches and sample the bsdf with weight one.
If the cosine of the final cache is bigger than 0.9 but less than 1, the outgoing direction of our path is sampled from the irradiance cache. It will however be converted to global and then to local coordinates at the intersection point in order to evaluate the bsdf at that point. Note that the sampled drection was represented in local cache coordinates at first and would also have been converted to global coordinates for a cosine of 1 anyway. See algorithm 1 on how caches are picked depending on their cosine to the surface normal at the current intersection point.

5.2.1 Multiple Importance Sampling of BSDFs and Caches

As implied in the previous segment, we will only actually combine bsdf and irradiance sampling whenever we hit a surface with a smooth bsdf and if a cache with a good enough cosine to the surface normal is available. In all other cases we continue our path by generating a sample from the bsdf.
We now have to do 2 things: Decide on a probability to select either bsdf or irradiance sampling, and decide how to compute the multiple importance sampling weights for each of them.

The Mitsuba Renderer, which was used as a framework to implement this work, offers the roughness of a bsdf in the form of a float value $\in [0, 1]$. A roughness value equal to ∞ is

used in addition to indicate perfectly diffuse surfaces.
We decided that even if a surface is completely diffuse, bsdf sampling should never be completely ignored. (TODO warum nochmal?) Then again we never want to pick irradiance sampling at perfectly specular surfaces for reasons explained above. So we decided on $\alpha = min(\text{roughness}, 0.9)$ as a threshold for randomly selecting bsdf or irradiance sampling.

Chapter (TODO one-sample model) explained how n exclusive sampling techniques can be combined with multiple importance sampling. In our case bsdf sampling is chosen with probability $1-\alpha$ and irradiance sampling with probability α. These probabilities obviously sum to one, they match c_1 and c_2 in equation (TODO).
Assume we generated a random number $> \alpha$ and sampled some direction ω with a probability p_{bsdf} from the bsdf. To compute that sample's MIS weight we also need to compute the probability p_{cache} for having sampled ω from the chosen cache (see chapter 5.1.2). The weighting function w_{bsdf} for the sampled direction can now be determined by the balance heuristic:

$$w_{bsdf} = \frac{p_{bsdf}}{p_{bsdf} + p_{cache}} \tag{5.1}$$

According to equation (TODO veach 9.15) the total weight we have to multiply to the energy carried along the path is

$$\frac{w_{bsdf}}{(1-\alpha) \cdot p_{bsdf}} \tag{5.2}$$

TODO wohin ist das $\frac{}{p_{bsdf}}$ verschwunden? ist das schon beim normalen Throughput-Gewichten mit dabei?

Assuming that the path's energy was already weighted similar to algorithm 3 with p_{bsdf} as $pdfValue$ before, we can ditch this part from the denominator and weight the path energy like this: (TODO stimmt das?)

$totalWeight \leftarrow \frac{p_{bsdf}}{(1-\alpha)\cdot(p_{bsdf}+p_{cache})}$
$pathEnergy \leftarrow pathEnergy \cdot totalWeight$

If the initial random number was $<= \alpha$ and 3 was already executed, the MIS weight for irradiance sampling can be computed accordingly:

$totalWeight \leftarrow \frac{p_{cache}}{\alpha\cdot(p_{bsdf}+p_{cache})}$
$pathEnergy \leftarrow pathEnergy \cdot totalWeight$

Literaturverzeichnis

[ED08] Thomas Engelhardt and Carsten Dachsbacher: *Octahedron environment maps.* In *Proceedings of Vision, Modeling, and Visualization 2008*, 2008.

[Kaj86] James T. Kajiya: *The rendering equation.* In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '86, pages 143–150, New York, NY, USA, 1986. ACM, ISBN 0-89791-196-2.

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Abänderungen entnommen wurde.

Karlsruhe, xx.xx.xx

.....

(**Max Mustermann**)