BACHELORARBEIT

Using Instancing to Model Boundary Conditions for Direct Flux in 3D High Performance TCAD Simulations

ausgeführt am Institut für Mikroelektronik
der Technischen Universität Wien

unter der Anleitung von
Dipl.-Ing. Dr.techn. Josef Weinbub, BSc
Dipl.-Ing.(FH) Paul Manstetten

durch

Dominik Koukola

MatNr. 1225419

December 22, 2016

Zusammenfassung

In den letzten Jahren werden beim Design von modernen Halbleiterbauelementen immer mehr dreidimensionale Layouts verwendet. Die Zahl der Probleme, die mit einer zweidimensionalen oder symmetrischen Simulation angenähert werden können, nimmt ab. In vielen Fällen ist eine komplette dreidimensionale Simulation notwendig um die Halbleiterbauelemente zu entwickeln. Dies erhöht den Druck auf die Rechenleistung der Simulationsmethoden noch weiter. Ein wichtiger Prozess in der Halbleiterherstellung ist das Ätzen. Ein wesentlicher Teil der Ätzsimulation ist die Berechnung der von einer Partikelquelle kommenden Flussraten auf der Oberfläche einer Geometrie. Diese Flussberechnung ist typischerweise der zeitaufwändigste Teil der Ätzsimulation. Diese Arbeit beschäftigt sich mit der direkten Flussratenberechnung einer dreidimensionalen Geometrie mit Intels Raytracing-Bibliothek Embree. Im Kontrast zum klassischen Ansatz, Strahlen an der Simulationsgrenze neu zu berechnen, werden die Randbedingungen durch Instanziierung, wie es in der Computergraphik verwendet wird, modelliert. Wir implementieren eine diffuse und verschiedene gerichtete Strahlenquellen um praktische Relevanz sicherzustellen. Um einen Richtwert für die Leistung zu geben, werden verschiedene Testfälle präsentiert und der Einfluss von Instanziierung, Strahlenquelle und Geometrie auf die Simulationszeit diskutiert. Zusätzlich wird das Rauschen der Flussraten, welches durch das Monte Carlo Sampling verursacht wird, analysiert. Die Ergebnisse zeigen, dass das Modellieren von Randbedingungen mittels Instanziierung ein attraktiver Ansatz ist, der eine erhebliche Beschleunigung der Ätzsimulation bietet.

Abstract

In the past years, the design of advanced semiconductor devices increasingly employs three-dimensional layouts. Situations where the problem of interest could be approximated with a two-dimensional or symmetric simulation are decreasing. In many cases, a full three-dimensional simulation is necessary to design the devices. This increases the pressure on the computational performance of the simulation methods even more. An important process in semiconductor fabrication is etching. An integral part of an etching simulation is the calculation of the flux rates on the surface of the geometry originating from a particle source. This flux calculation is typically the most time consuming part in the etching simulation. This thesis focuses on the calculation of the direct (primary) flux rates of a three-dimensional geometry using Intel's ray tracing library Embree. The boundary conditions are modeled using instancing as used in computer graphics in contrast to the classic approach to recalculate the rays at the simulation boundary. We implement a diffuse and various directed ray sources to ensure practical relevance. To give an indication for the performance, different test cases are presented and the influence of instancing, ray source and geometry on the simulation time are discussed. Additionally, the noise of the flux rates due to Monte Carlo sampling is analyzed. The results show that using instancing to model boundary conditions is an attractive approach that offers substantial speed-up of the etching simulation.

Acronyms

TCAD	Technology Computer-Aided Design
3D	Three-Dimensional
MC	Monte Carlo
VTK	Visualization Toolkit
API	Application Programming Interface
BVH	Bounding Volume Hierarchy
ISA	Instruction Set Architecture
OpenMP	Open Multi-Processing

Contents

1 Introduction

In recent years the miniaturization of semiconductor technology is approaching physical boundaries leading to more complex structures. Therefore, sophisticated semiconductor product development uses numerical models to analyze and predict process and device characteristics. This field of modeling is referred to as Technology Computer-Aided Design (TCAD) and is widely used in the semiconductor industry by engineers in the design process. TCAD tools allow it to get a deeper understanding of the technology and lower costs of development and manufacturing [1].

In the beginning of TCAD, simulations were limited to one- or two-dimensional models due to the lack of computing power and memory. With the progress in processing capabilities of computer hardware, three-dimensional (3D) TCAD was made available, making it possible to better analyze and understand technology through simulation. On the other hand, the vast advances in the semiconductor industry and the increasingly complex structures are demanding a full 3D simulation as two-dimensional representations give insufficient results and symmetry approximations are often not possible [2].

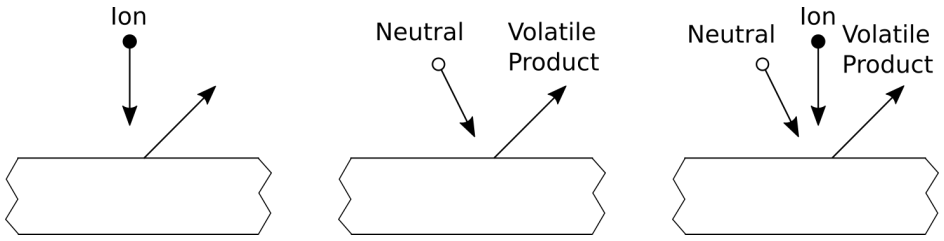
Today's 3D TCAD applications have high accuracy requirements and are large scale, demanding a lot of computing power, thus slowing down the development process. To keep the design time at a reasonable level, simulation time needs to be decreased while accuracy and resolution has to be at least maintained.

Within TCAD, process simulation is used to predict the device fabrication steps. The thus generated device is then used to determine the electrical characteristics via device simulations [2]. In turn, the characteristics are forwarded to circuit simulations to predict the behavior of an ensemble of devices. This work operates within the field of process simulation, as such as the here presented findings will allow to accelerate parts of the essential surface evolution simulations required for, e.g., etching simulations [3].

1.1 Plasma Etching - Physical Process

A key part in the fabrication of modern semiconductor devices is plasma etching as it allows efficiently to produce anisotropic etch profiles with a high yield. The general concept of plasma etching is that chemically reactive plasma discharges are used to alter the surface of materials. There are various kinds of etching processes that differ in directionality, selectivity, etch rate and yield [4].

Some commonly used plasma processes to remove material from surfaces are illustrated in Figure 1.1 [5]. Sputtering, shown in Figure 1.1a, is the process of atom ejection due to bombardment with energetic ions. Chemical plasma etching (Figure 1.1b) uses gas-phase neutral atoms supplied by the plasma discharge that react with the surface material forming volatile etch products and leading to the removal of material from the surface. Ion enhanced etching, illustrated in Figure 1.1c, is the combination of both, chemical reactions with neutral atoms and bombardment with ions [6].



(a) sputtering (b) chemical (c) ion enhanced

Figure 1.1: Plasma etching processes

1.2 Plasma Etching - Simulation

The process of plasma etching can be modeled as ballistic transport of particles that hit the geometry surface. This causes a reaction at the surface element and depending on the materials leads to a local etchrate on the surface. This evolving of the surface is modeled in timesteps as illustrated in Figure 1.2. Due to the assumption of ballistic sources, the particles can be represented by rays allowing similar ray tracing methods as are used in computer graphics [7]. The rays emitted by the sources are referred to as primary or direct rays. In the process of plasma etching the particles can be reemitted after hitting the surface. These are modeled as reflections and are referred to as secondary rays. As the surface changes, a new flux calculation is performed for each timestep, making it very time consuming as the flux calculation is the bottleneck of the simulation [8].

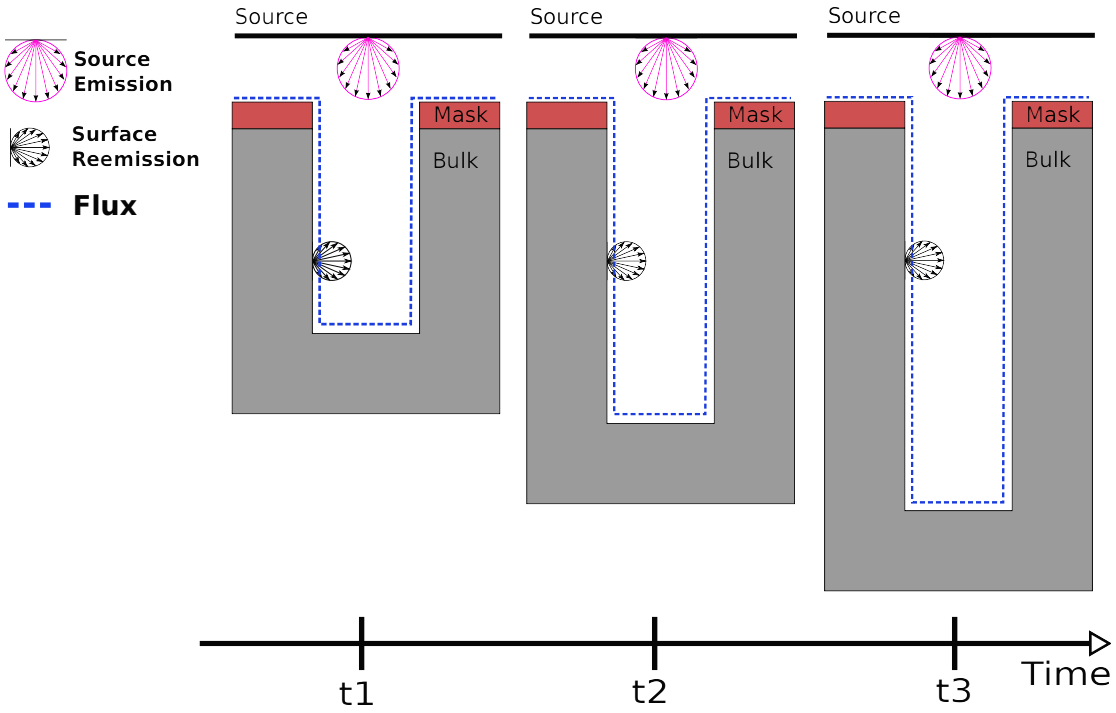


Figure 1.2: Schematic of three geometries (mask and bulk) corresponding to three timesteps in an etching simulation. A diffuse planar source is indicated with the bold line and the rays at the top. The dotted line indicates the area where the flux has to be computed.

In Figure 1.3, the simulation flow is illustrated. After the setup of ray sources, geometry, and material models, a loop is entered where first the flux for all surface elements is calculated. With this information and the material model, the reaction of the surface elements is calculated. After that, the surface representation is updated and the loop starts again. For each timestep the loop is run through and after a specified number of steps the final surface representation is the result [9].

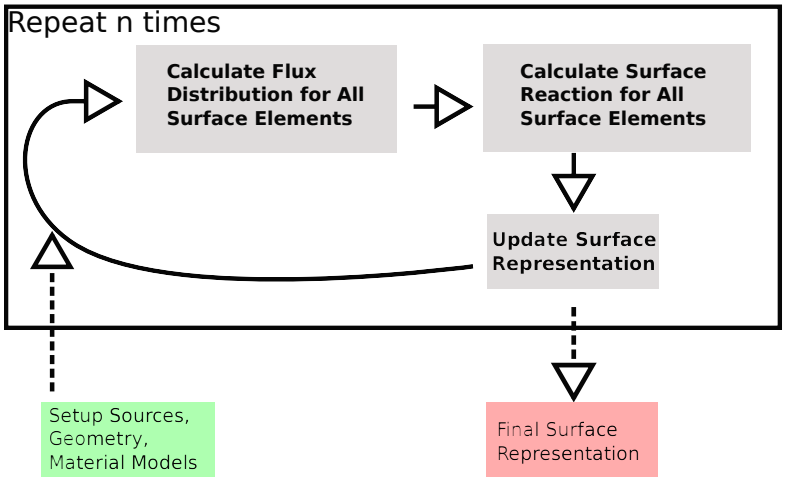


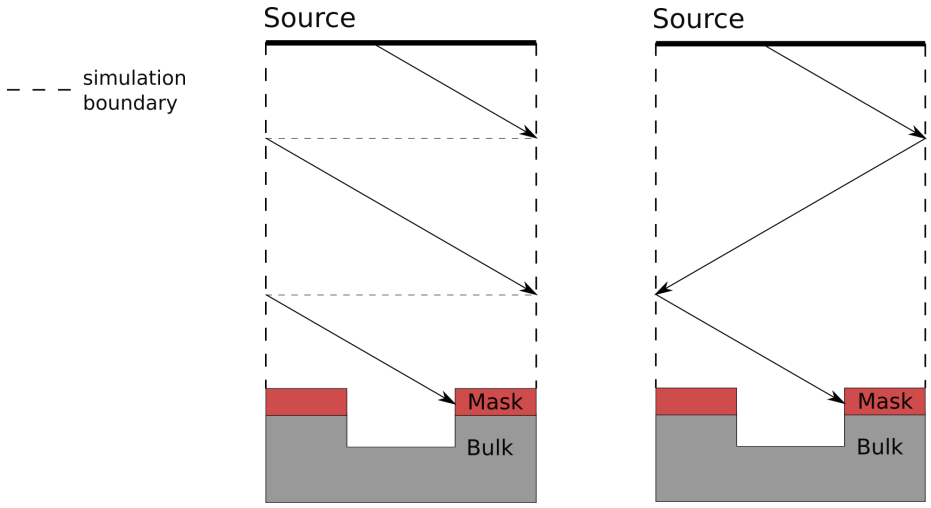
Figure 1.3: Simulation of an etching process. Starting with an initial setup, the main loop calculates the flux distribution and surface reaction for all surface elements, and updates the surface representation. After n time steps, the final surface representation is generated.

1.3 Geometric Boundary Conditions

Geometric boundary conditions describe what happens when a ray exits the boundaries of the simulation domain. The two general types are the reflective and the periodic boundary conditions. With the periodic boundary condition, the rays are reemitted with the same direction on the opposite side of the domain, as illustrated in Figure 1.4a. With the reflective boundary condition, illustrated in Figure 1.4b, the rays are reflected at the boundary.

The classical approach is to recalculate the ray direction and position at each intersection with the boundary. Instead of this recalculation, this thesis evaluates an approach to use *instancing* to model the boundaries of the domain. The initial geometry is extended by placing instances according to the boundary conditions. The rays, which all start above the initial geometry, are traced against the extended geometry and do not have to be recalculated at the intersection with the boundary.

To enable an instancing approach for reflective and periodic boundary conditions the simulation boundary box has to be rectangular. Otherwise, depending on the geometry, only one of the boundary conditions or none can be modeled using instancing. The instancing approach is described in more detail in Section 2.2.



(a) Periodic (b) Reflective

Figure 1.4: Different boundary conditions. a) Periodic: Exiting rays reenter domain at the opposite side of the domain with same direction. b) Reflective: Exiting rays are reflected into the domain.

Furthermore, the surface is defined to be explicitly represented as a triangle mesh. Generally a surface can be represented implicitly, typically using a level set, or explicitly, where a common choice is a triangle mesh [10]. To be able to utilize the implementations of modern ray tracing frameworks (e.g. Embree [11]) we use a triangle mesh in this work.

The source is modeled as a plane above the surface. The secondary rays are not considered in this thesis, only the direct (primary) flux is simulated.

With this definition the input parameters of this problem are:

- A triangulated mesh of the surface with a rectangular boundary box
- The boundary condition
- The ray source and the number of rays

The output of the simulation is:

- The direct flux rate for each triangle

1.4 Outline of the Thesis

In Section 2, the implementation of the ray sources is described and the approach to model boundary conditions using instancing is introduced and explained. Intel's ray tracing library Embree and the Visualization Toolkit (VTK) along with the main functions from these frameworks used in this thesis are described in Section 3. Section 4 presents and discusses the ray distribution and the direct flux results. The consequences of noise on the direct flux rates are analyzed. The impact of instancing, different ray sources and number of triangles on the performance of the simulation is discussed in Section 5. The summary in Section 6 gives an overview over the main results of this thesis.

2 Methods

In this section the main methods are described. First, different ray sets are implemented to be able to simulate reasonable test cases. Second, it is described how instancing can be used to model the periodic and reflective boundary conditions. Finally, the direct flux calculation is explained and the absolute, relative and normalized flux rates are defined.

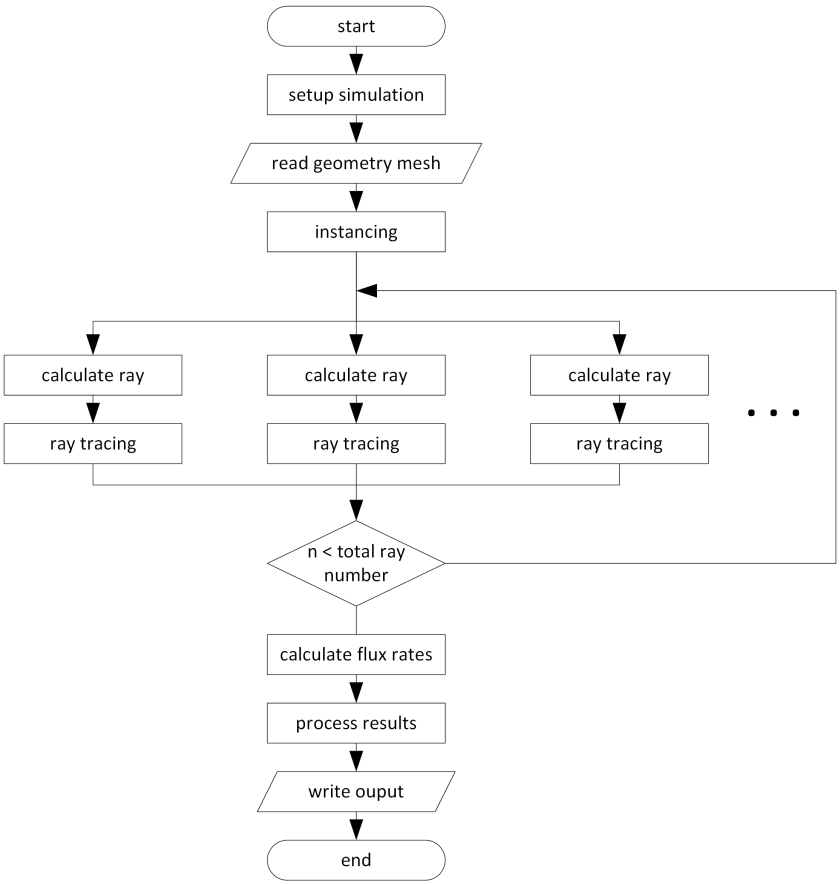


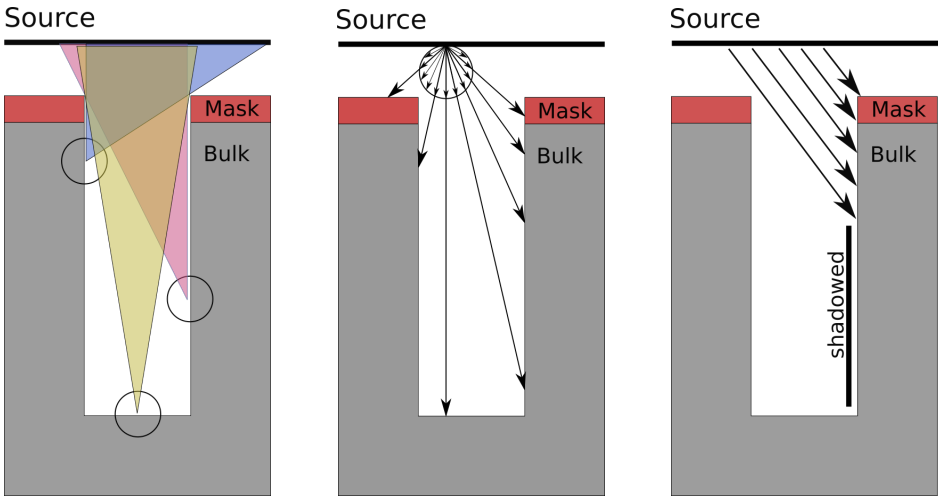
Figure 2.1: Algorithm for direct flux simulation. Ray calculation and tracing can be done in parallel as indicated in loop.

To give an overview, the simulation algorithm is presented in Figure 2.1. The first step of the simulation is to read the simulation setup and the geometry mesh. That includes information about boundary condition, ray source, number of rays and the size and form of the instance extension. After importing the triangulated geometry mesh, transformation and rotation matrices for instancing are computed and the scene with all instances is built. Next the time consuming simulation loop is entered. For each loop run the origin and the direction of a ray are determined. A ray tracing algorithm is used to calculate if and where the ray hits the geometry mesh. The information of ray hits is later used for the flux calculation. Depending on the available hardware and methods used the ray tracing loop can be run sequentially or using multiple compute units in parallel. When all rays are processed the direct flux rates are calculated and optionally the results can be processed further (e.g. statistical analysis).

2.1 Ray Source Modeling

In general, there are two approaches to get the flux contribution of the source towards each surface element: a) The bottom-up approach shown in Figure 2.2a and b) the top-down approach illustrated in Figure 2.2b and 2.2c.

In the bottom-up approach the visibility of the sources is viewed from the perspective of the surface element and the flux is determined by integration over the solid angle in which the source is visible.



(a) Bottom-up (b) Top-down with diffuse source (c) Top-down with directed source

Figure 2.2: Different methods and different source types for direct flux calculation. a) In the bottom-up approach, for each surface point, the source contribution is integrated over the visible solid angle. b) In the top-down approach, the source is sampled using a MC approach and the contributions are detected on the surface elements. c) When the source is highly directed, the flux rates change abruptly when entering a shadowed region.

The accuracy can be controlled locally by changing the sample resolution of the direction. The top-down approach, as used in this thesis, is a MC technique that samples the angular distribution of the source using a high number of rays.

2.1.1 Monte Carlo Sampling

With MC sampling the areal source is sampled with a large number of rays. The trajectory of the rays has to be calculated and the first surface intersection has to be determined. The direct flux is determined by the number of hits the surface element gets and the accuracy is controlled by the total number of rays. The emission characteristics of the source determine the origins and directions of the rays. As MC sampling is a numerical approximation there is a noise that depends on the number of rays and on the spatial resolution of the geometry. A higher number of rays reduces the noise, whereas a higher number of triangles increases it [12]. The effect of the noise on the direct flux rate is presented in Section 4.3.

2.1.2 Ray Sets

To be able to simulate a variety of problems a couple of different ray sets were analyzed: a parallel ray set, a diffuse ray set and a power cosine ray set. The rays were implemented as areal sources where the origins are placed randomly with a uniform distribution across the source plane. The diffuse and power cosine ray set were additionally implemented as a point source where all rays share the same origin. As we assume a rectangular simulation boundary box the source plane is a rectangular area as big as the geometry's horizontal outline. The position of the source plane is usually close to the highest point of the geometry so that the number of required instances is kept at a minimum.

In the following, we define the vertical direction to be the z-axis and the horizontal plane to coincide with the xy-plane.

Parallel Ray Set

The parallel ray set is straight forward. All rays have the same direction and the origins are distributed as described above. An illustration of this ray set is given in Figure 2.2c where all emitted rays have the same direction. it also shows the effect of shadowing where due the parallel direction some areas cannot be reached by the rays.

Diffuse Ray Set

In the diffuse ray sets the rays are emitted in every direction with equal probability. In Figure 2.2b, the diffuse ray set is illustrated originating from one point of the surface. The implementation of this is the same problem as picking a random point on a surface of a sphere. George Marsaglia provided a method to solve this problem using two independent uniform distributions to create a single uniform distribution on the surface of a unit sphere [13]. This method was used for the analysis.

Formally the diffuse and also the power cosine ray set can be described with the cumulative distribution function F or the probability density function f. We define θ to be the polar angle (angle between the z-axis and the ray) and φ to be the azimuth angle (angle between the xy-projection of the ray and the x-axis). Then, the distribution of the diffuse ray set regarding θ is:

$$F_\theta(\theta) = 1 - cos(\theta) \tag{2.1}$$
$$f_\theta(\theta) = sin(\theta) \tag{2.2}$$

The distribution regarding ϕ is uniform.

Power Cosine Ray Set

The power cosine ray set has a uniform distribution regarding φ, but has a different more directed distribution regarding θ:

$$F_\theta(\theta) = 1 - cos(\theta)^{n+1} \tag{2.3}$$
$$f_\theta(\theta) = (n+1) \cdot sin(\theta) \cdot cos(\theta)^n \tag{2.4}$$

The variable n is a positive integer. A higher value means the distribution is more focused. To implement the power cosine ray set an algorithm derived by [14] was used.

2.1.3 Ray Set Distribution Test

To evaluate the distribution of the diffuse and power cosine ray sets a test case regarding θ and one regarding ϕ were implemented. In both test cases, the origin of all rays is the point with the coordinates (0,0,-1) and a triangulated disk at $z = 0$ with the center in the axis origin was used as a geometry to detect the ray hits.

Theta (θ) Distribution Test

Figure 2.3 shows the projection of the rays originated in (0,0,-1) onto the xy-plane with the corresponding angle and radius relation.

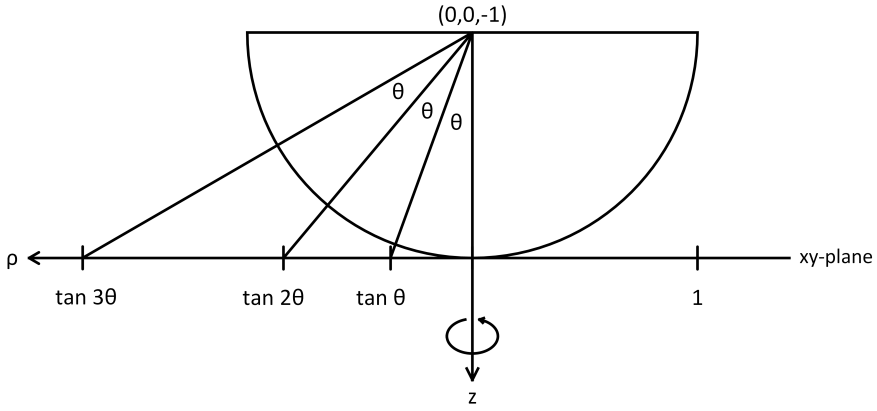


Figure 2.3: Structure of the θ distribution test. The source origin is on the z-axis at distance 1 above the xy-plane. Rays emitted with the polar angle θ hit the plane at radius $\rho = tan(\theta)$.

The cumulative distribution function can be calculated as the number of hits within the radius $\rho \leq tan(\theta)$ divided by the total number of rays:

$$F(\theta) = \frac{\text{hits}(\rho \leq tan(\theta))}{\text{total number of rays}} \tag{2.5}$$

With this relation the distributions of the implemented ray sets can be compared to their corresponding cumulative distribution functions. As a triangulated mesh is used the circles are not perfectly circular and the resolution is limited due to the size and number of triangles.

Phi (φ) Distribution Test

To test the uniform distribution regarding φ the circular plane was cut into k same sized circular arcs as shown in Figure 2.4. As the rays are uniformly distributed regarding φ the number of hits of each circular arc is expected to be the same.

$$F_k(i) = \frac{\text{hits in circular arc i}}{\text{total number of rays}} = \frac{1}{k}, \text{with } k \in \mathbb{N}, i = 1, .., k \tag{2.6}$$

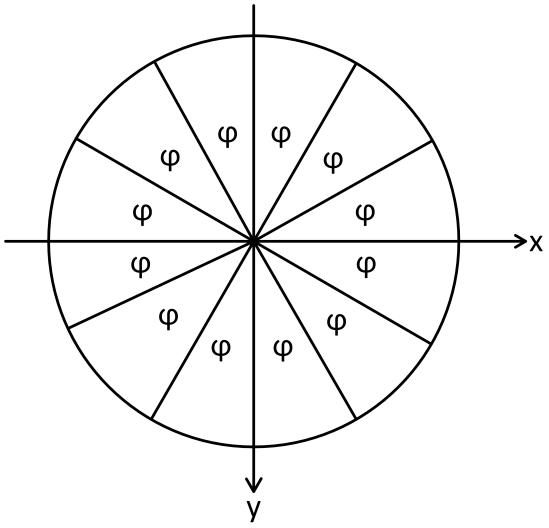


Figure 2.4: Structure of the φ distribution test. The disk is cut into same sized circular arcs with the angle φ.

The results of these two test cases for the diffuse and power cosine ray set are discussed in Section 4.

2.2 Instancing

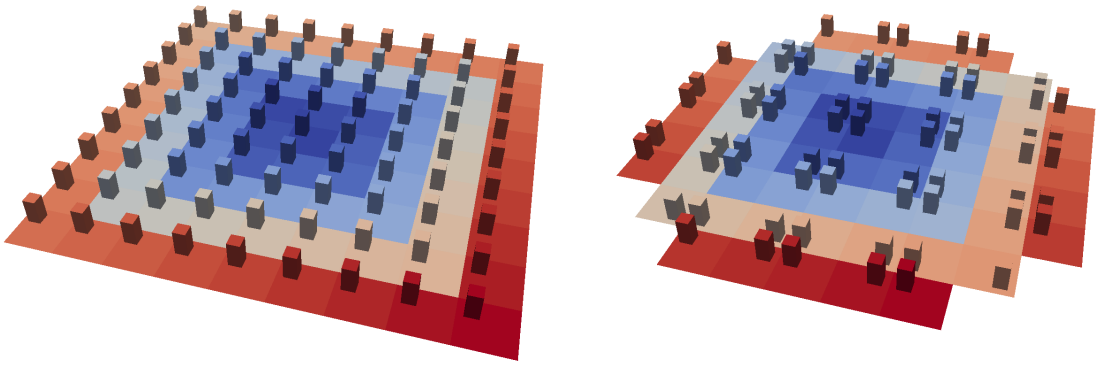
Instancing is a method commonly used in computer graphics to make copies, so called instances, of an object without actually copying an object's data but by just providing a transformation and rotation matrix [15].

To model boundary conditions using instancing, instances of the geometry are put next to each other. Depending on the boundary condition they are either periodic or mirror symmetric towards each other. Using this extension of the domain, rays that exit the original domain do not need to be recalculated as they are traced further till they hit an instance of the geometry. The geometry with all it's instances form a scene. The source plane is still limited to the boundary box of the geometry as extending the source plane would model a different problem than the initial one. Every instance of a geometry triangle contributes to it's flux rate as the resulting flux rate is the sum over all its instances.

The two approaches that were analyzed to extend the scene with instances are: a) A rectangular extension where rectangular rings of instances are added and b) a circular extension where a radius determines how many instances are made. The number instances required depends on the distribution of the rays and the form of the geometry. A focused beam, for example, will require less instances then a diffuse source.

Figure 2.5 gives an illustration of the resulting scene of a asymmetric pole geometry using rectangular extension with a periodic boundary condition and using circular extension with a reflective boundary condition.

As we assume a rectangular boundary, putting instances of the geometry and their mirrored representation next to each other is straight forward for reflective boundary conditions. For periodic boundary condition opposite sides of the geometry have to fit together. This can easily be extended to model the periodic boundary condition with any periodic structure that has this characteristic. For a reflective boundary condition using instancing is limited to a few mirror symmetric extendable boundaries of geometries (e.g. rectangle or isosceles right triangle).



(a) Periodic with rectangular extension (b) Reflective with circular extension

Figure 2.5: Instancing with a asymmetric one pole geometry that has a continuous periodic boundary. Colors indicate the identifier (ID) of the instance used in the implementation.

2.3 Direct Flux Calculation

In the simulation loop, the number of hits for each triangle is computed. The flux rate is defined by number of rays per area. As the triangles have different area sizes the number of hits does not represent the flux rate. Therefore, first the area is calculated with Equation 2.7 using vector analysis from [16].

$$\textit{triangle area} = \frac{1}{2}\|\overrightarrow{AB} \times \overrightarrow{AC}\| \tag{2.7}$$

Figure 2.6 illustrates a cutout of a triangulated mesh with a triangle represented by the vertices A, B and C. It's area is highlighted in gray.

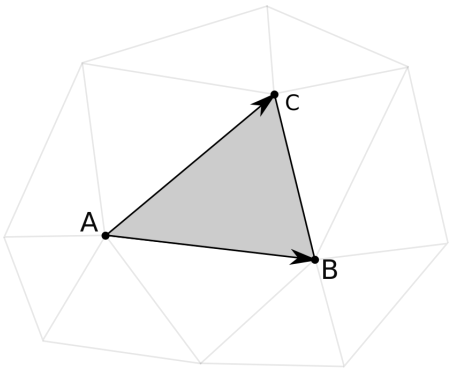


Figure 2.6: Triangulated surface representation

In this thesis we distinguish between the absolute, a relative and a normalized direct flux rate:

$$\text{absolute} = \frac{\text{hits}}{\text{triangle area}} \tag{2.8}$$

$$\tag{2.9}$$

$$\text{relative} = \frac{\text{hits}}{\text{triangle area}} \quad \frac{1}{\text{total number of rays}} \tag{2.10}$$

$$\tag{2.11}$$

$$\text{normalized} = \frac{\text{hits}}{\text{triangle area}} \quad \frac{1}{\text{planar element flux}} \tag{2.12}$$

The planar element flux is defined as the absolute flux that a planar area element would receive when it is fully exposed to the source.

3 Tools

For instancing and ray tracing, we used the open source ray tracing framework Embree [11], in contrast to, e.g., NVIDIA's ray tracing engine OptiX [17], which is not open source. For data exchange and visualization we used the open source VTK [18].

3.1 Instancing and Ray Tracing with Embree

Embree is an at Intel developed collection of high performance ray tracing kernels [11] that are optimized for the x86 architecture. It provides highly optimized low level kernels for Bounding Volume Hierarchy (BVH) construction, ray traversal and ray-triangle intersection. The kernels can be accessed within an application by a high level Application Programming Interface (API) allowing it to be used with minimal programming effort. The implementation provides optimization for different Instruction Set Architecture (ISA) [19]. In the following, the utilized key functions provided by the Embree library are introduced. Their names start with `rtc`, which stands for `r`ay `t`racing `c`ore.

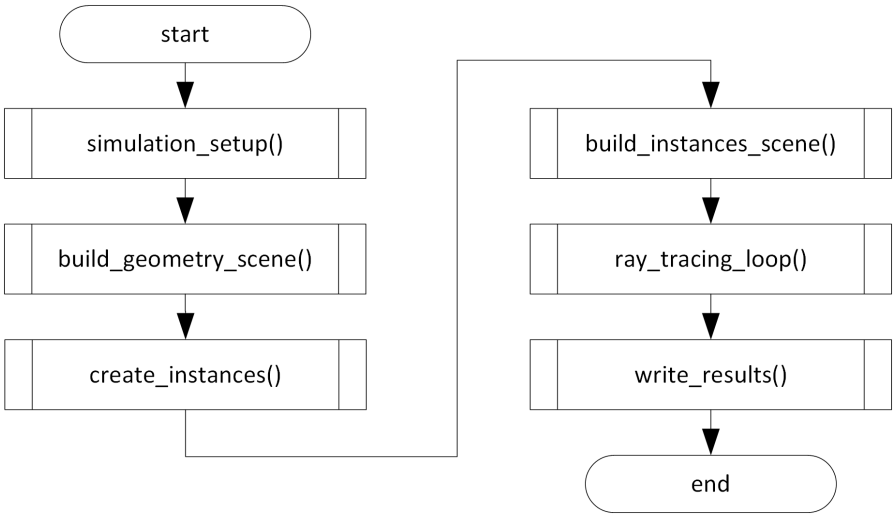


Figure 3.1: Structure of the implementation of the simulation algorithm.

The implemented simulation algorithm starts, as shown in Figure 3.1, with the setup of the simulation. That includes the import of simulation information as boundary condition, ray source type and number of rays, and the calculations of transformation and rotation matrices for instancing. After that, the scene for the geometry is built as in Listing 3.1.

```
1  int main ( int argc , char * argv [])
2  {
3    //Setup with variable declarations and initialization
4    ...
5
6    //Register operation recommended by Embree
7    _MM_SET_FLUSH_ZERO_MODE(_MM_FLUSH_ZERO_ON);
8    _MM_SET_DENORMALS_ZERO_MODE(_MM_DENORMALS_ZERO_ON);
9
10   //Create a device
11   RTCDevice device = rtcNewDevice(..);
12
13   //Create container for geometry
14   RTCScene scene0 = rtcDeviceNewScene(device, ..);
15
16   //Add a triangulated mesh object
17   unsigned geomID = rtcNewTriangleMesh(scene0, ..);
18
19   //Fill array of vertex positions
20   Vertex *v = (Vertex *)rtcMapBuffer(scene0, geomID, ..);
21   fillVertexBuffer(..);
22   rtcUnmapBuffer(scene0, geomID, RTC_VERTEX_BUFFER);
23
24   //Fill array of triangle vertices
25   Triangle *t = (Triangle *)rtcMapBuffer(scene0, geomID, ..);
26   fillTriangleBuffer(..);
27   rtcUnmapBuffer(scene0, geomID, RTC_INDEX_BUFFER);
28
29   //Indicate that scene description is finished
30   rtcCommit(scene0);
```

Listing 3.1: Creating an Embree scene using a triangle mesh and triggering the generation of the BVH structure using the `rtcCommit()` call (Line 30).

The scene, a container for the triangulated geometry, is created and filled with the geometry data. The `rtcDeviceNewScene()` (Line 14) call creates a container for the surface geometry. Depending on the flags set it can be among others optimized for a static or dynamic scene and for coherent or incoherent rays. After filling the scene with information `rtcCommit()` (Line 30) triggers internal data structure building operations. These operations are already parallelized internally by Embree.

As next step, a new scene for the geometry and all its instances is created (Listing 3.2). For each instance an instance object of the geometry stored in `scene0` is created in `scene`. After it is transformed accordingly the kernel is notified about a geometry change before the scene is committed.

```
31   //Create container for instances
32   RTCScene scene = rtcDeviceNewScene(device, ..);
33
34   for(int i = 0; i < numberOfInstances; i++)
35   {
36     //Add instance object of scene0 in scene
37     unsigned int instance = rtcNewInstance(scene, scene0);
38
39     //Transform instance
40     rtcSetTransform(scene, instance, .., transformationMatrix[i]);
41
42     //Inform that geometry has changed
43     rtcUpdate(scene, instance);
44   }
45
46   rtcCommit(scene);
47
48   //Compute ray tracing for all rays
49   rayTracingLoop(..);
50
51   //Calculate flux rates and write results
52   writeResults(..);
53
54   //Destroy scenes, device and its contents
55   rtcDeleteScene(scene0);
56   rtcDeleteScene(scene);
57   rtcDeleteDevice(device);
58
59   return 0;
60 }
```

Listing 3.2: Creating an Embree scene using transformed instances of an existing scene. The `rayTracingLoop()` (Line 49) is shown seperately in Listing 3.3.

The parallelizable ray tracing loop (Line 49) is shown in Listing 3.3. In each iteration a ray is generated and intersected with the scene. Embrees `rtcIntersect()` call (Line 9 in Listing 3.3) delivers the results by modifying the ray structure `ray`. This information is stored to calculate the flux rates. With the determined hit information the flux rates are calculated.

The `rtcIntersect()` call (Line 9), that delivers the closest hit of the ray segment with the scene, supports parallelization: It is up to the user to parallelize the ray tracing loop, which can be implemented via, for instance, a straight forward parallel for loop construct (cf. Section 5.3).

```
1  void rayTracingLoop(..)
2  {
3    for(int i = 0; i < totalNumberOfRays; i++)
4    {
5      //Generate a ray according the simulation information
6      RTCRay ray = getRay(..);
7
8      //Find closest hit of a ray segment with the scene
9      rtcIntersect(scene, ray);
10
11     //Store intersection information for later processing
12     storeHitInfo(ray);
13   }
14 }
```

Listing 3.3: Ray tracing loop.

The most important parameters of the `RTCRay` datastructure are:

`Vec3f org`:	ray origin
`Vec3f dir`:	ray direction
`float tnear`:	start of ray segment
`float tfar`:	end of ray segment, set to hit distance after intersection
`int geomID`:	ID of hit geometry
`int primID`:	ID of hit primitive (triangle ID)
`int instID`:	ID of hit instance

The rays are defined as ray segments, which means that they have a start and end point. `Vec3f` is a structure of three `float` variables. Embree delivers the information at what distance on the ray segment which triangle in which instance of which geometry is hit. For flux calculation only the triangle ID `primID` is used as all instances contribute towards the flux rate and only one geometry is used.

Two features of Embree that could improve the simulation speed [20], but are not analyzed in this thesis are:

- Intersect a bundle of rays at once using the functions `RTCintersect4`, `RTCintersect8` or `RTCintersect16` (which functions are supported depends on the ISA).
- The distinction between `RTC_SCENE_COHERENT` and `RTC_SCENE_INCOHERENT` rays during scene building.

3.2 Visualization and Debugging with VTK

VTK is an open-source toolkit for 3D computer graphics, image processing, and visualization [18] that began as a companion software to [21] and is now a general-purpose system used in numerous applications. It provides, among other features, a variety of data representations, readers and writers to exchange data with other applications [22]. Some of them were used in this thesis for read and write operations of triangulated geometry and ray data stored in *obj* and *vtp* files. An example on how a triangulated mesh was read from an *obj* is Listing 3.4.

```
vtkio::TriangleMesh readobjmesh(std::string const &FileName)
{
  //Initialize reader
  vtkSmartPointer<vtkOBJReader> reader = vtkSmartPointer<
   vtkOBJReader>::New();
  reader->SetFileName(FileName.c_str());
  reader->Update();

  //Get file data
  vtkPolyData* polydata = reader->GetOutput();

  //Get information about triangle mesh size
  int numpoints = polydata->GetNumberOfPoints();
  int numtriangles = polydata->GetNumberOfCells();

  //Create triangle mesh object with corresponding size
  vtkio::TriangleMesh mesh = vtkio::TriangleMesh(numpoints,
   numtriangles);

  //Fill the mesh object with the read data
  fillMeshWithReadData(mesh, polydata);

  return mesh;
}
```

Listing 3.4: Read geometry data from obj file using VTK.

ParaView

ParaView is a data analysis and visualization application [23] based on VTK [24] developed to deal with extremely large datasets [25]. It is used in this thesis to visualize the geometry, instances, rays and the calculated direct flux rates for debugging purposes. ParaView provides a number of filter and plotting functions to analyze the resulting flux rates.

4 Simulation Results

In this section, the results of the ray set distribution tests are discussed in Section 4.1. An example problem for the direct flux rates is presented in Section 4.2 and the noise due to the MC sampling is analyzed in Section 4.3.

4.1 Ray Set Distribution

In the following, the results of the two test cases introduced in Section 2.1.3, to evaluate the ray set distribution, are presented for a power cosine ray set with the degree of $n = 2$ for 10^4 rays and 10^6 rays.

Theta (θ) Distribution

In Figure 4.1, the cumulative distribution and probability density functions of the ray sets theta distribution for 10^4 rays, 10^6 rays and the theoretical values as defined in Section 2.1.3 are presented. As the cumulative distribution functions show almost no visible difference only the theoretical graph is shown. The density function for 10^4 rays has visible fluctuations regarding the expected function. The density function for 10^6 rays shows only few deviations.

This validates that the implemented ray set follows the theoretical θ distribution. The fluctuation is explained by the effect of MC-sampling described in Section 2.1.1. A higher number of rays lets the graphs approach the theoretical curves, whereas a higher θ resolution increases the fluctuations. The density function naturally has higher fluctuations as it represents the slope of the cumulative function.

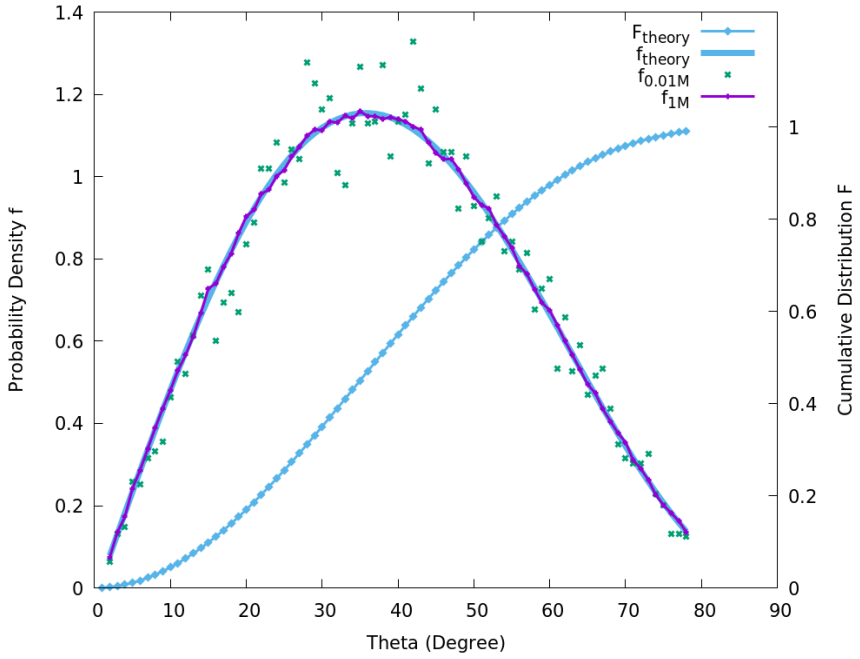


Figure 4.1: θ cumulative distribution $F(\theta)$ and probability density $f(\theta)$ of a power cosine source with $n = 2$

Phi (φ) Distribution

As the φ distribution is expected to be constant, the difference to the theoretical value of each slice was calculated. The resolution was chosen to be $\Delta\phi = 2°$, which means, that the circle was divided into 180 slices. Figure 4.2 shows the histogram of the deviations of the number of rays in each ϕ-slice. The x-axis indicates the difference to the theoretical value in percent and the y-axis relates to the number of slices with the respective difference.

To provide quantities information for the fluctuations we introduce the mean μ and the standard deviation σ. Let Δn be the error in per cent and k the total number of slices then μ and σ are calculated for this discrete set as followed:

$$\mu = \frac{\sum_{i=1}^{k} \Delta n_i}{k} \tag{4.1}$$

$$\sigma = \sqrt{\frac{1}{k} \sum_{i=1}^{k} (\Delta n_i - \mu)^2} \tag{4.2}$$

The area under the curve represents the number of slices which is the same for all three. The test with 10^6 rays shows the most fluctuations ($\sigma = 1.40\%$), the test with 10^7 rays shows less ($\sigma = 0.43\%$) and the test with 10^8 rays shows the least fluctuations ($\sigma = 0.16\%$). All three are inside of 5% deviations to the theoretical value.

The implemented ray set follows the expected theoretical distribution, but contains a noise due to the MC-sampling. The question of how many rays are necessary for a particular resolution of the direct flux simulation is discussed in Section 4.3.

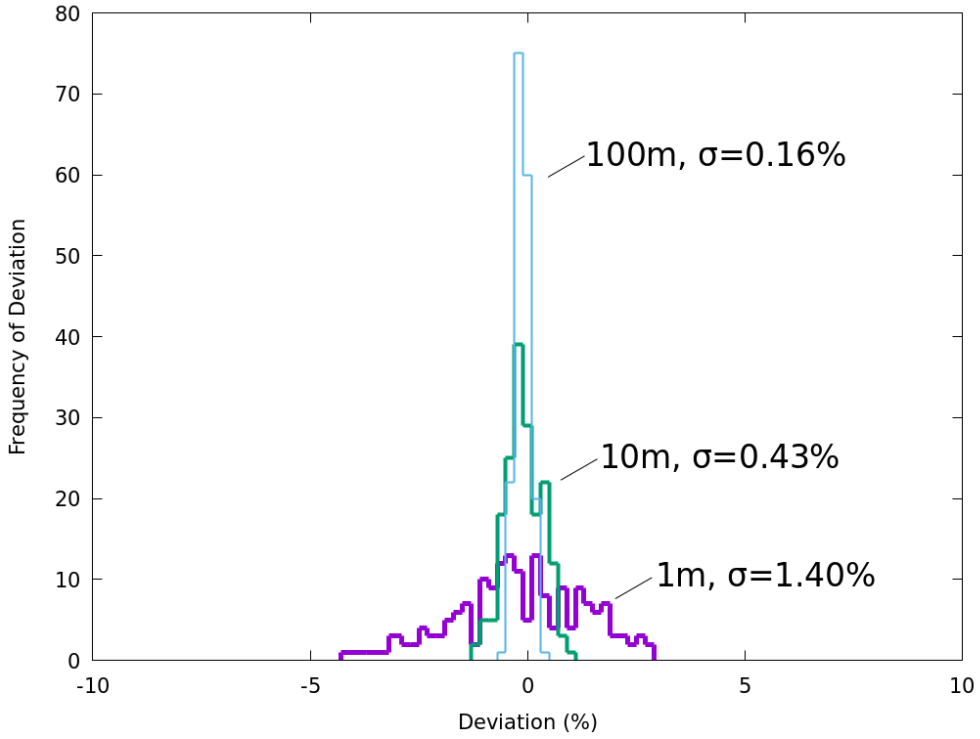


Figure 4.2: φ deviation distribution of a power cosine source with $n = 2$

4.2 Direct Flux Rates

As an example problem for the direct flux simulation we introduce an asymmetric *tower* geometry of about 50k triangles. The tower is in the center of the left upper quadrant of a 2x2 square plane and is 1 length unit high as shown in Figure 4.3 and Figure 4.4.

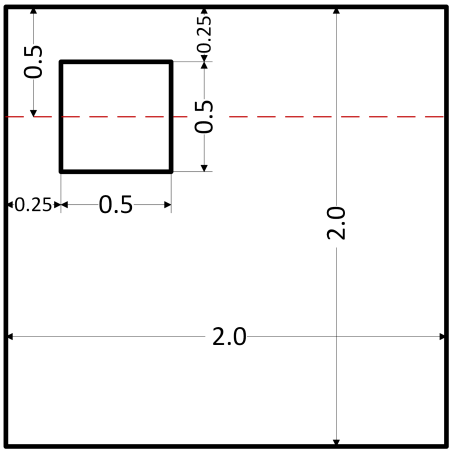


Figure 4.3: Topview of the asymmetric tower geometry

This geometry was simulated with a power cosine ray set with a degree of $n = 2$ with 10^8 rays. The surface source was set 0.1 length units above the highest point of the geometry as shown in Figure 4.4.

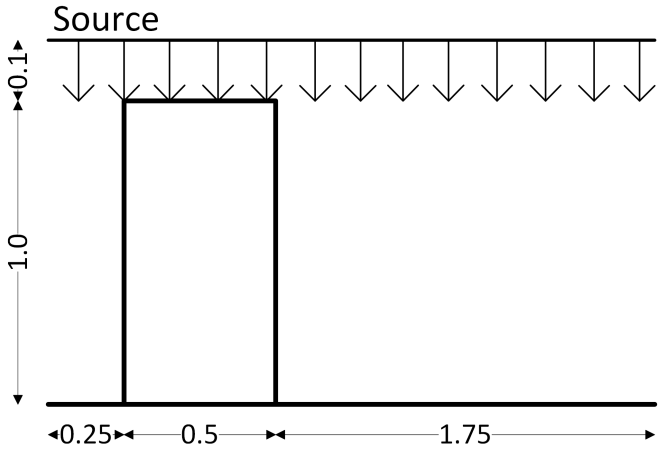


Figure 4.4: Sideview of the asymmetric tower geometry and simulation setup

The simulation is run once without instancing, once with a reflective and once with a periodic boundary condition. The extension of the instances for the boundary conditions is circular with a radius of $r = 20$. The resulting direct flux rate is evaluated as a cut along the $y = -0.5$ line of the geometry. The line is marked in Figure 4.3 as a dashed line.

Figure 4.5 shows the normalized flux rates along the interface, which stem from cutting the domain with a plane at $y = -0.5$ from left to right. The projected length is the length the line runs through. At the bottom and the top it is the x-length and at the wall it is the z-length.

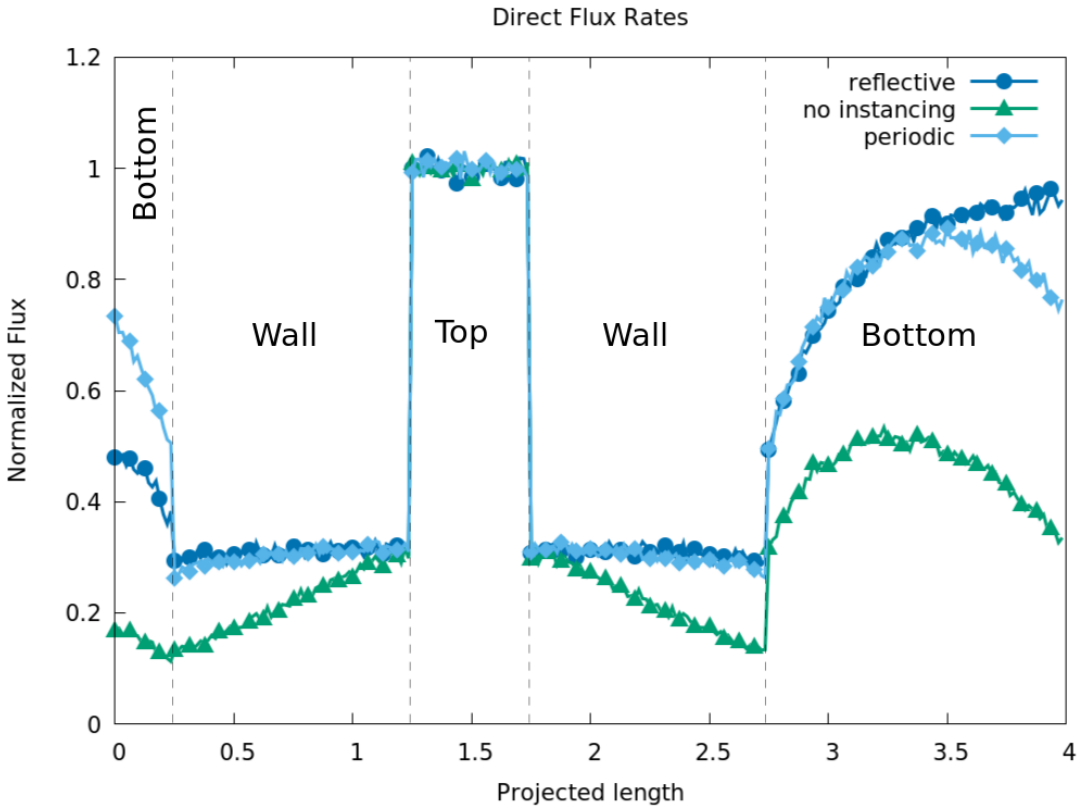


Figure 4.5: Direct flux rates along the $y = -0.5$ slice with a power cosine ray set with a degree of $n = 2$ with 10^8 rays and a circular extension with an radius $r = 20$ for the reflective and periodic boundary condition.

All three have a normalized flux rate of about 1 at the top, but at the walls and at the bottom the simulation with no instancing has lower rates. The simulations with instancing have similar rates at the walls and at the top, but differ in the area around the edges at the bottom. On the left side at the bottom the periodic boundary condition causes the highest rate whereas on the right side the reflective causes the highest rate. Both the rates with periodic boundary condition and the result without instancing show a decline towards the edge on the right side. At the wall the rate with no instancing shows a significant decline approaching the bottom. The two rates with instancing show only a little decrease whereas the one with periodic boundary condition has a slightly bigger one.

The simulation with no instancing has a lower normalized flux rate at the bottom and at the wall as rays that exit the boundary are not taken into account. The rate at the top is similar to that of the other two as rays that exit the boundary contribute only very few to the rate at the top. The decline at the wall is due to the increasing spatial angle towards the bottom. With instancing this effect is very small for this geometry. The difference at the edges of the two results with instancing is because of the different shadows the tower causes depending on the boundary condition.

4.3 Noise

As described in Section 2.1.1 the MC-sampling causes unwanted fluctuations of the direct flux rate. To analyze this noise for an example problem we introduce a rotationally symmetrical cylinder geometry as shown in Figure 4.6.

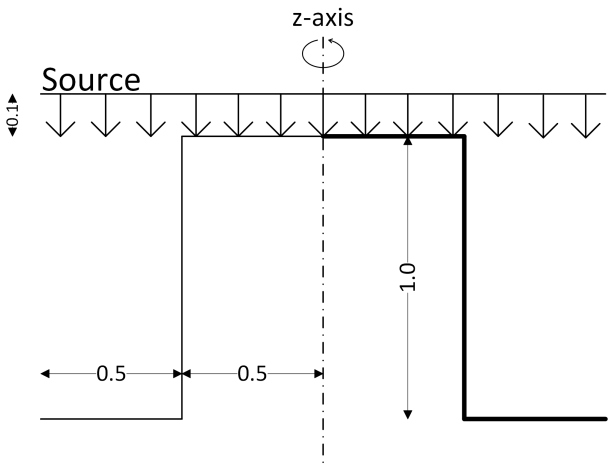


Figure 4.6: Cylinder geometry

The cylinder is placed in the center of a 2x2 square plane. It is 1 length unit high and has a diameter of 1. For the simulation the ray source is again placed 0.1 length units above the highest point of the geometry. As it is a surface geometry it can be used as a hole as well as a pole by simple flipping it upside down.

For the simulation, the geometry was cut into rotationally symmetrical bins as illustrated in Figure 4.7. The bins on the bottom and top plane form anuli and the bins at the wall form cylinder rings. The light areas at the corners of the big plane are not evaluated in the following as they are not rotationally symmetrical. The width of the bins around two to three triangles wide so that there is a representative number of triangles in a bin, but the averaging effect is not too big.

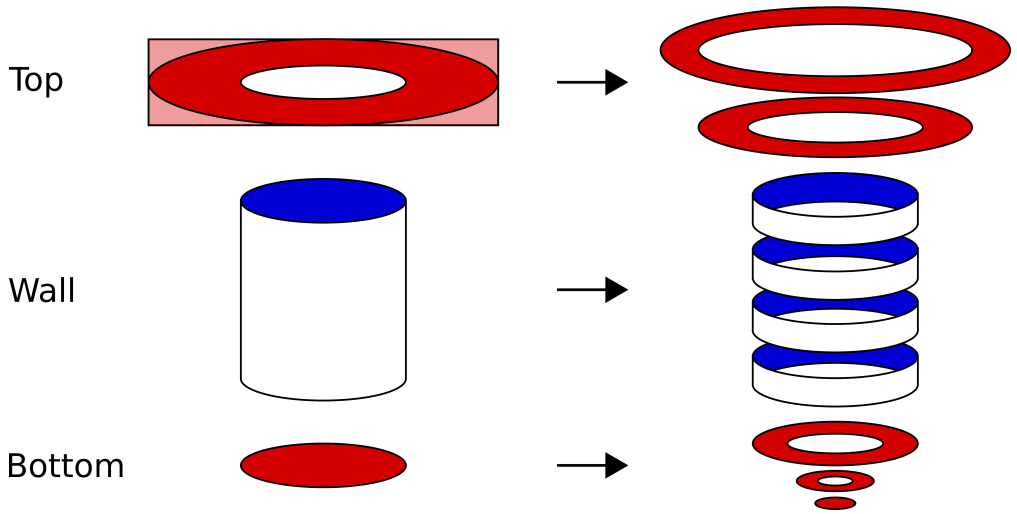


Figure 4.7: Slicing the cylinder surface geometry into rotationally symmetrical bins creating anuli at the planes and cylinder rings at the wall

Each triangle in a bin is expected to have the same flux if the ray source is as well rotationally symmetrical around the rotation axis of the geometry, but due to the noise of the ray source there are fluctuations in the bins. In the following the normalized flux rates of these bins are plotted against the projected length. The projected length is the length of the bold line in Figure 4.6 and runs through and represents the width of the bins. At the flat planes it represents the radius of the anuli and at the wall the height of the cylinder rings. It always starts at the center of the geometry and thus at the anuli with the smallest radius. It then goes up from the bottom when it is a hole or down from the top when it is a pole and ends at the anuli with the biggest radius.

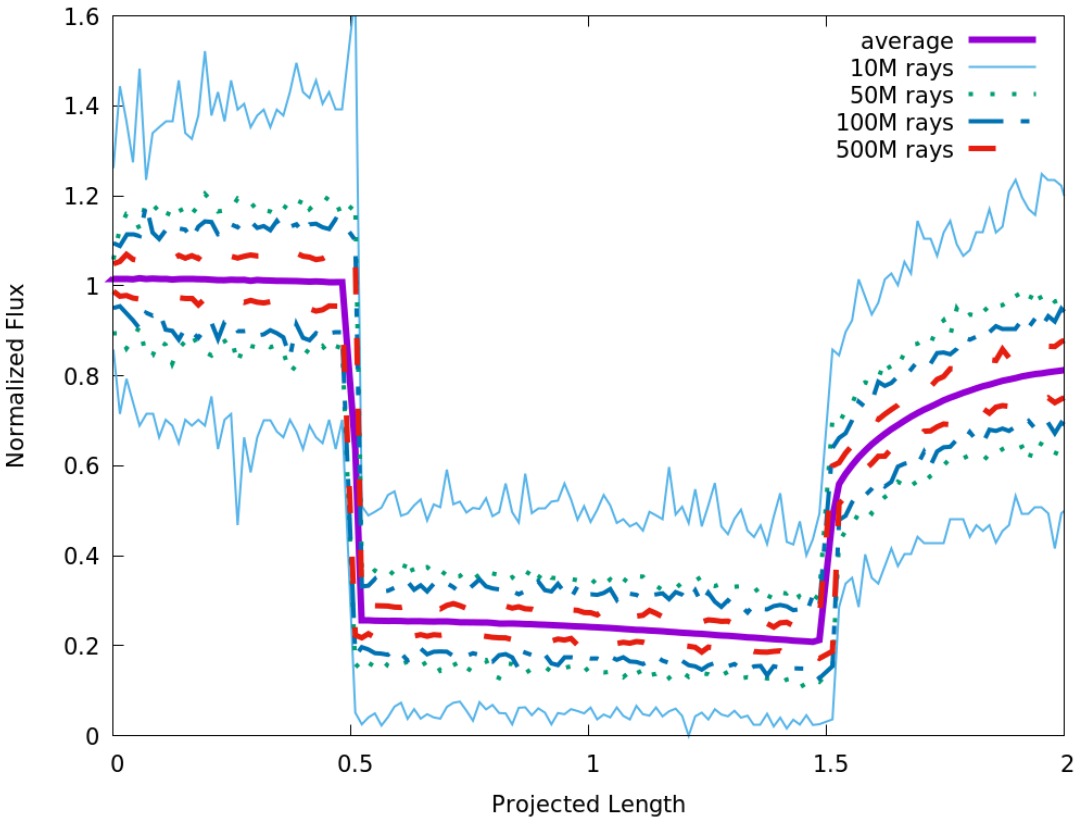


Figure 4.8: Noise of the cylinder geometry as a pole with around 260k triangles for a power cosine ray set with a degree of $n = 2$ and a circular extension of instances with radius $r = 20$. For 10^7, $5 \cdot 10^7$, 10^8 and $5 \cdot 10^8$ rays.

Figure 4.8 shows the normalized flux rate for the cylinder geometry as a pole with around 260k triangles for 10^7, $5 \cdot 10^7$, 10^8 and $5 \cdot 10^8$ rays. The x-axis represents the bold line from Figure 4.6. The solid graph represents the flux rate of the bins and the other lines represent the maximum and minimum flux rate of a triangle in the respective bin for the different number of rays. The normalized flux of the bins is for all four figures almost the same. Therefore only one average is shown in Figure 4.8. It is almost steady with 1 at the top, has a little decline at the wall towards the bottom and rises at the bottom plane towards the edge. The maximum and minimum rates of triangles inside the bins vary the most with 10^7 rays and decrease with a rising number of rays. The steady flux at the top is because the ray source hits the whole top equally.

On the wall is a much lower flux because of the directed distribution of the power cosine source. The slight decrease of the flux towards the bottom of the wall is explained with the decreasing visible solid angle from a surface point towards the source. The rise at the bottom towards the edge is because the farther away from the pile the more of the ray source is visible for the triangles. The normalized flux rates of the bins almost doesn't vary with the number of rays because the bins contain many triangles thus forming an average over the triangles and reducing the noise. However, the fluctuations of a single triangle inside the bins is still very high with 10^7 rays. By increasing the number of rays the noise of the triangle can be decreased. With $5 \cdot 10^8$ rays the maximum and minimum flux rate deviations inside the bins are already very low. The minimum flux rate deviations depends on the problem being analyzed.

Figure 4.9 shows the effect of increasing the number of triangles of the cylinder geometry as a hole on the noise of the flux rate.

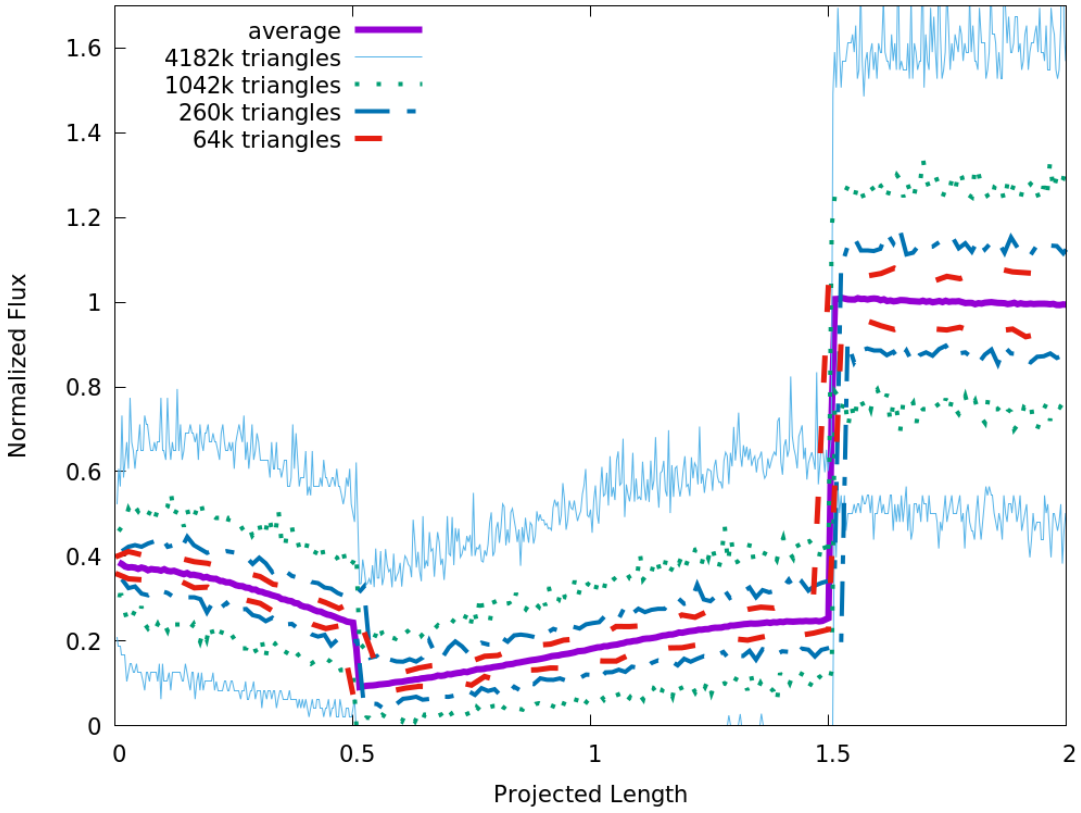


Figure 4.9: Noise of the cylinder geometry as a hole with 10^8 rays and with 64k, 260k, 1042k and 4182k triangles.

The x-axis again represents the bold line from Figure 4.6 but this time the geometry is flipped upside down. The solid line, that again represents the normalized flux rates of the bins is almost the same for all four, therefore only one average is shown. The maximum and minimum rates of a triangles in the bins, however, increase with the number of triangles. The hole shows a steady flux rate at the top, a decreasing rate at the wall towards the bottom and an increase at the bottom towards the middle.

The flux rate at the top is steady as the rays hit the top equally, because there are no shadows. The flux rates on the bottom and on the wall is due to the solid angle in which the source is visible. The horizontal plane sees a higher amount of the source then the vertical wall. The behavior of the noise is as expected. A higher number of triangles increases the resolution of the geometry and demands a higher number of rays to keep the noise steady.

For an etch simulation, it is necessary that the fluctuations are in a given range to provide a certain statistical accuracy [7]. Figure 4.10 shows the maximum flux deviations of triangles inside bins of the hole geometry with 260k triangles depending on the number of rays.

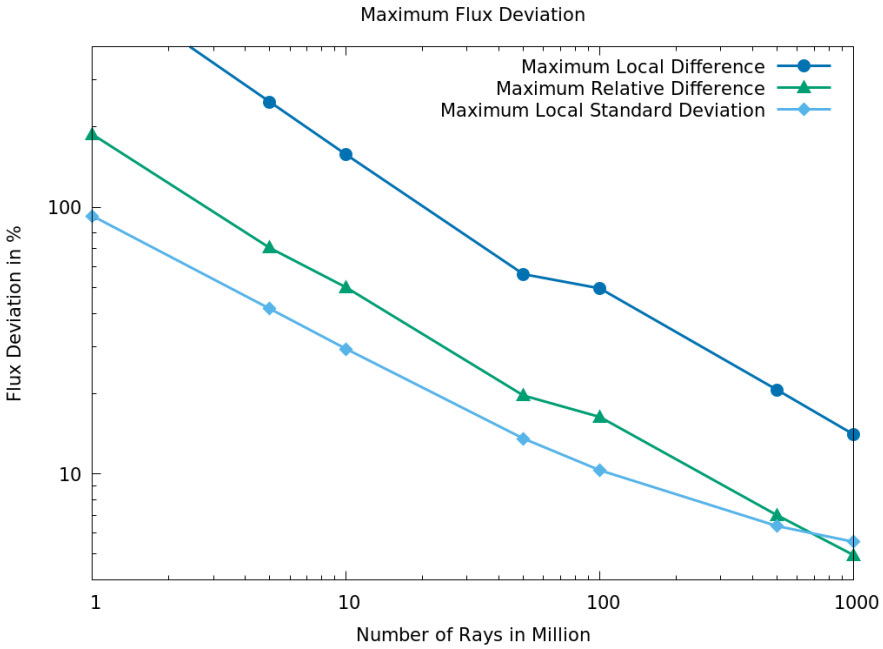


Figure 4.10: Maximum flux deviations of the hole geometry with 260k triangles

The maximum local difference is the highest deviation of any triangle's flux rate to its local bin flux rate. A less stricter criteria is the maximum relative difference, which is the highest deviation of any triangle's flux rate relative to the maximum bin flux rate. Additionally, as another measurement for noise, the standard deviation for each bin relative to its mean flux rate was analyzed and the maximum relative standard deviation of all bins is shown in Figure 4.10. To be able to visualize a broader range of rays the scale is double logarithmic.

The maximum difference graphs decrease roughly linearly with the number of rays. For this simulation setup they are approximately proportional to each other. Both are way over 100% when using 10^6 ray, but decrease steadily and at 10^9 rays the maximum local difference is around 14% and the maximum relative difference is around 6%.

Some potential reasons for the remaining noise and errors in the simulations, which provide future research directions, are:

- One reason why the fluctuation doesn't reach zero could be the use of a single precision floating point representation, as it is used by Embree during the ray tracing. This could cause some triangles being hit more often then their neighboring triangles leading to a fluctuation that would not decrease with a higher number of rays.
- Another reason could be the size and position of the bins. When the flux rate has a high slope across the bin width, triangles on one border have a different flux rate than triangles on the other border, leading to noise inside the bin that doesn't decrease with a higher number of rays. Therefore, the bin width has to be chosen small enough. An extreme example for this are the bins of the edges which can contain triangles from the higher flux horizontal region and the lower flux vertical region. This leads to a fluctuation inside the bin that doesn't decrease with an increasing number of rays.
- A third reason for this could be a bad ray distribution. When the random number generator used for the ray generation doesn't have a perfect uniform distribution. This could lead to artificial noise in the source distribution and consequently also artificial noise in the flux rates of the triangles.
- Another possibility is that the geometry is not perfectly rotationally symmetrical. When, for example, some triangles inside a bin are more inclined then others, they can have a different visible solid angle of the source, which would lead to a remaining noise. The angle of inclination with respect to the ray source has a big impact on the flux rate.

5 Performance Results

In this section, the performance of the intersection loop is analyzed. In Section 5.1 an indicator for the performance is given. The effect of source properties and of the number of triangles are discussed. The cost of instancing for power cosine rays, and for vertical and horizontal traversing rays is investigated in Section 5.2. Results of parallelizing the intersection loop with OpenMP are presented in Section 5.3.

The simulations were run in a virtual machine with 6 GB main memory on an Intel Core i5-4210U processor (2 cores).

5.1 Ray Tracing Performance

We use the cylinder geometry introduced in Section 4.3 to analyze the ray tracing performance. The ray source placement is identical to Figure 4.6. The setup was simulated for various numbers of triangles and using two ray sets: a parallel ray set, and a power cosine ray set with $n = 2$. The time of the intersection loop and the time to generate the rays were measured.

The performance for both ray sets declines with an increasing number of triangles. The performance for the parallel ray set decreases faster. When comparing the performances without the ray generation (solid lines in Figure 5.1) the parallel ray set is higher at first but with more triangles the power cosine ray set has a better performance than the parallel ray set. This could be explained by the fact that all parallel rays hit the geometry, but only around 72% of the power cosine source do. Embree uses the BVH in any case, but if the rays do not even enter the topmost bounding box the tracing ends right away. This is, of course, computationally cheaper than a deep traversal of the bounding box hierarchy.

The time how long the ray is traversed through the bounding boxes depends on the origin and direction of the ray. When the geometry is made of less triangles The parallel ray set has without ray generation a higher performance than the power cosine set, even though more rays hit the geometry. The reason could be that tracing for parallel ray hits is easier than for hits from the power cosine ray set.

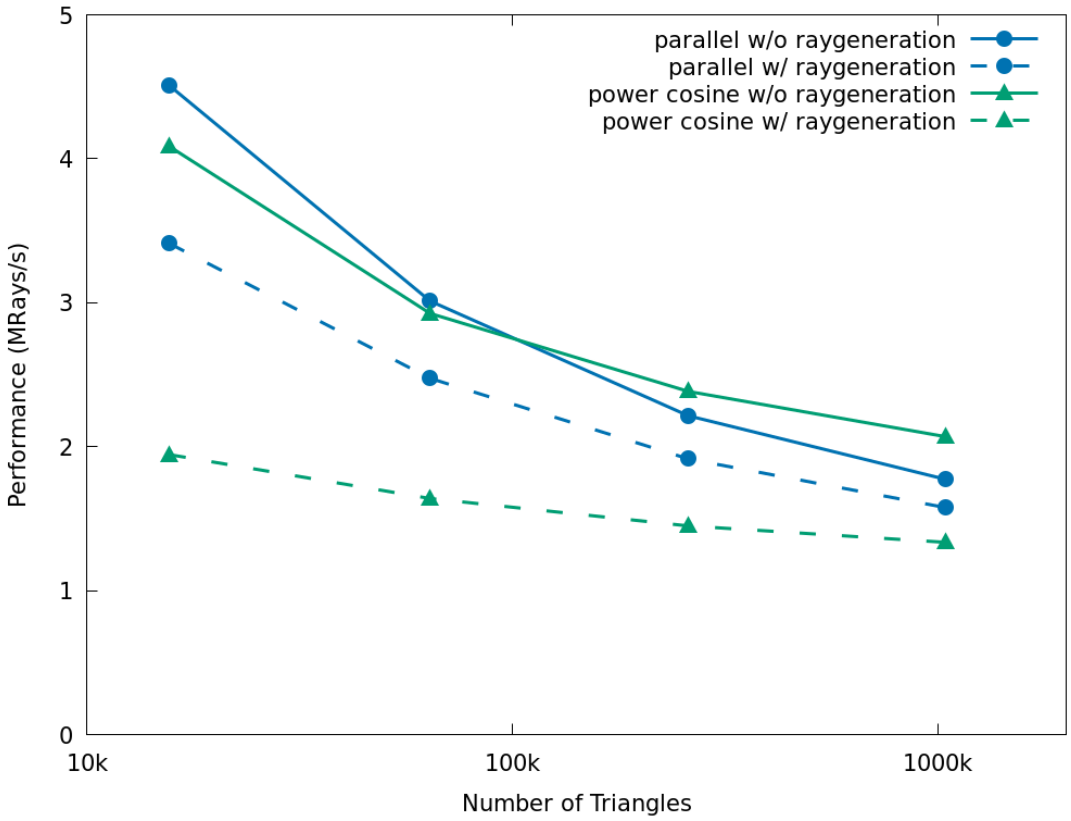


Figure 5.1: Performance of the intersection loop with and without ray-generation for a parallel and a power cosine $n = 2$ ray source. The cylinder surface (Figure 4.6) is used as geometry (without instancing). For the parallel ray set 100% and for the power cosine ray set around 72% rays hit the geometry.

The performance graphs for the parallel ray source are closer to each other because its ray generation is less computationally intense. The reason for the general decline is that the BVH structure gets bigger when the number of triangles increase. Thus the ray tracing for rays that hit the geometry needs more time.

5.2 Cost of Instancing

The performance of the intersection loop with ray-generation in dependence of the number of instances is analyzed for power cosine rays, and vertical and horizontal traversing rays. The instances are extended circularly.

5.2.1 Power Cosine Source

The cost of instancing is analyzed for the same cylinder geometry as in Section 5.1, for a power cosine $n = 2$ ray set. The results for different numbers of triangles are shown in Figure 5.2.

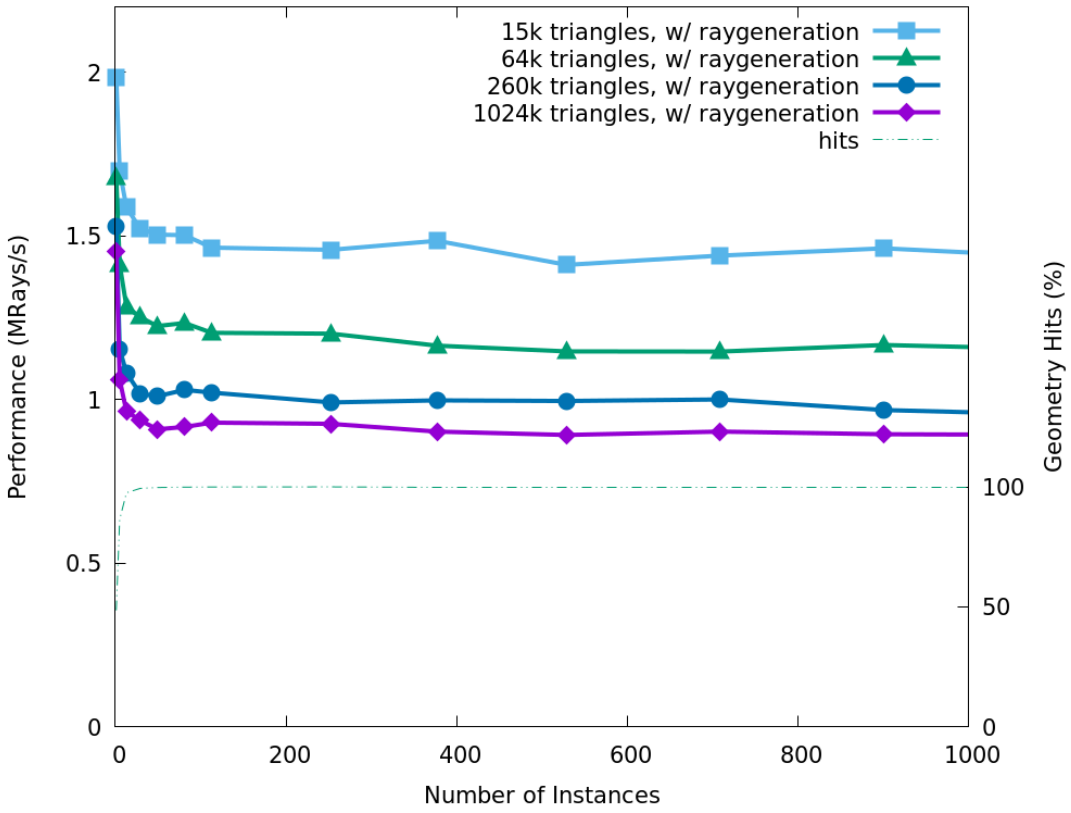


Figure 5.2: Influence of the number of instances on the ray tracing performance when using a power cosine $n = 2$ ray set on the cylinder geometry (Figure 4.6). The source domain is the same size as the geometry domain.

For all four the performance drops with the number of hits. This is plausible as rays that hit the scene are more computationally intense than *nohits*. At around 100 instances more than 99,9% of rays hit the scene. A further increase of the number of instances does not change the performance of the intersection loop. This is as expected. There should be no overhead as it is assumed that Embree is optimized and doesn't overlap top level bounding boxes.

The drop of the performance from 0 instances to over 100 instances is for 15k triangles about -27%, for 64k about -31% and for 260k about -36%. For 1024k triangles it is -38%.

5.2.2 Vertical Traversal

The influence of instancing for vertical traversal is analyzed for the tower geometry that is introduced in Section 4.2. The problems were analyzed for a varying number of instances that are extended circularly. For the vertical traversal a parallel ray source was used with rays parallel to the z-axis. The position of the source is identical to the setup in Figure 4.4.

Figure 5.3 shows the results of the simulation for: a) The ray source domain is the same as the geometry domain (2x2) and b) the dimensions of the source domain are 50x50. The solid lines represent the performance of the loop with ray-generation and the dashed lines represent the number of geometry hits.

For a) there is a drop of performance by about -27%. This is unexpected as the number of hits stays steady. All rays hit the initial geometry. A steady performance independent of the number of instances would be expected for an optimal ray tracing algorithm.

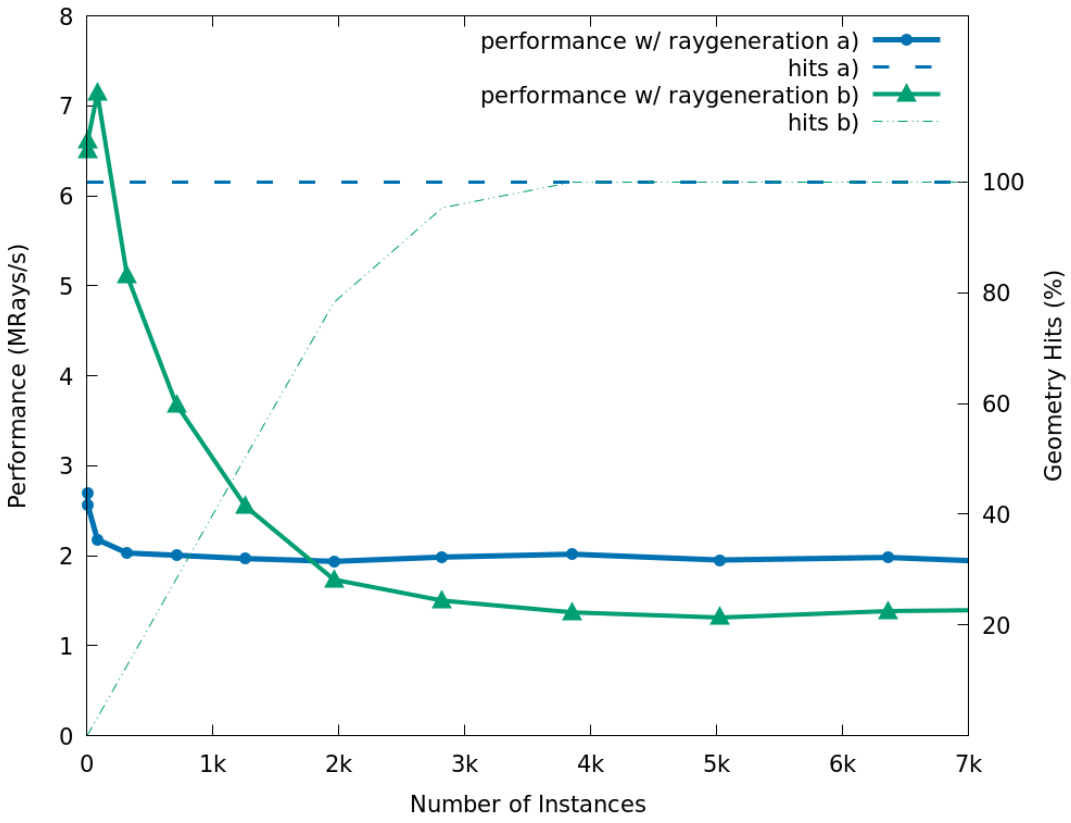


Figure 5.3: Influence of the number of instances on the ray tracing performance when using a parallel ray set on the tower geometry, as in Figure 4.3. The ray source domain is for a) identical to the geometry domain (2x2) and for b) with the dimensions 50x50.

Concerning the increase of the domain to 50x50 - b): The scene under the ray source growths with an increasing number and with around 4k instances surpasses the size of the ray source. The time of the intersection loop increases until the instances cover the full area under the source. When all rays hit the instanced geometry an increase of instances does not effect the performance. However, the resulting performance for the loop is about 25% lower than of a) in Figure 5.3.

5.2.3 Horizontal Traversal

To test the performance for horizontal traversal the same geometry as for vertical traversal (Section 5.2.2) was used. The source was, similar to a lighthouse, a point source that emits horizontally in all directions. It was placed at half height of the tower at the z-axis of the geometry. The result is shown in Figure 5.4.

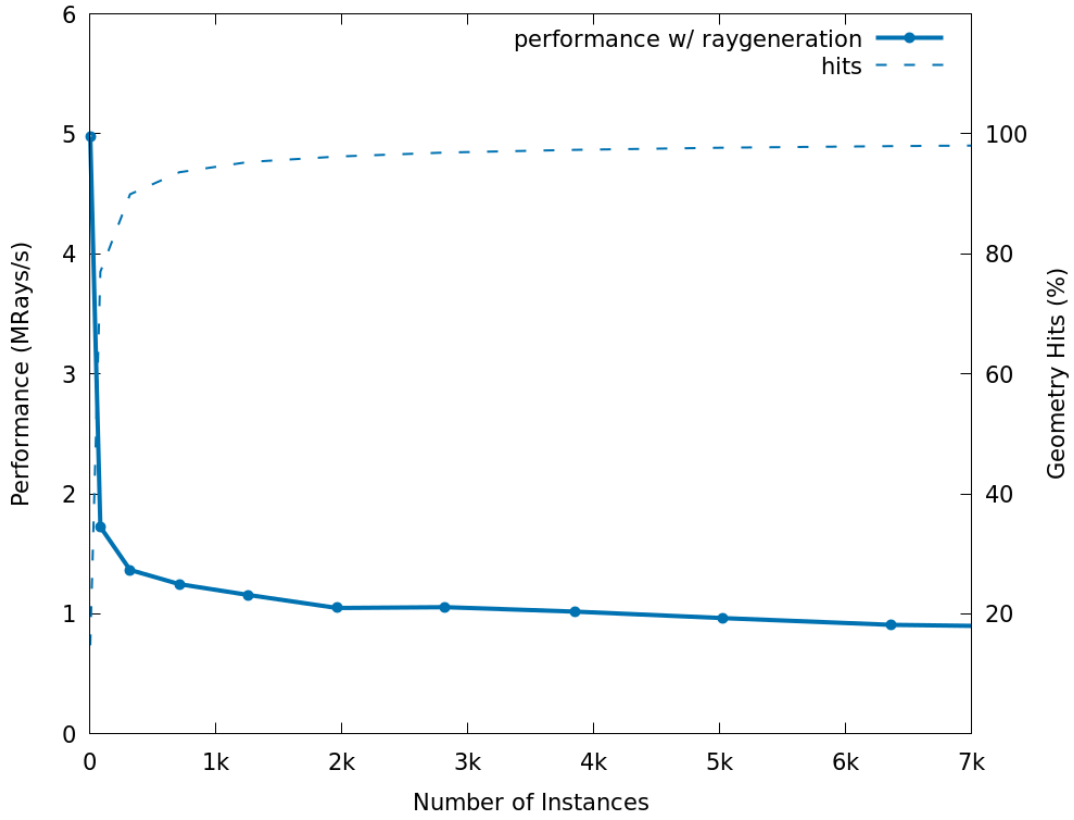


Figure 5.4: Intersection loop performance dependence of the number of instances for the tower geometry (Figure 4.3) with a horizontal lighthouse source at 0.5 height at the z-axis.

The time for the intersection loop is similar as the result for the vertical test. The performance of the intersection loop decreases with the number of hits. In the beginning there is drop as most rays hit one of the nearest 1k instances. After that there is still a steady decrease in performance with a rising number of instances. This could be because there are still rays that even with a high number of instances do not hit the geometry. These rays have to be traversed through a high number of instances.

5.3 OpenMP

The intersection loop was parallelized with OpenMP. To show that it is possible to trace rays in parallel, the tower geometry was used with the setup as described in Figure 4.4. For this simulation vertical rays with starting points placed on a regular xy-grid were used.

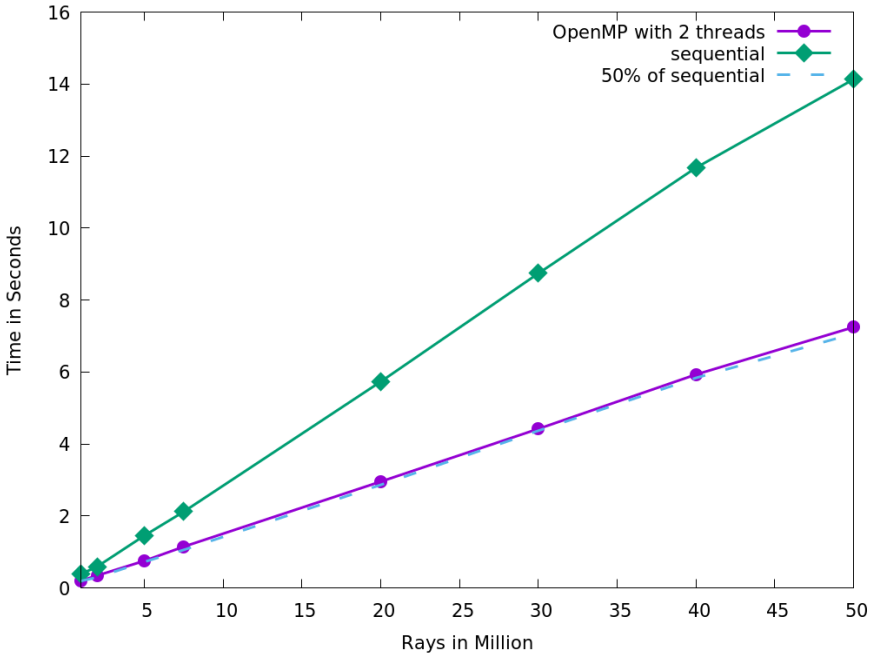


Figure 5.5: Time of the intersection loop with and without OpenMP with about 50k triangle tower geometry. Sequential performance is around $3.5 \cdot 10^6$ and parallel performance is about $6.8 \cdot 10^6$ rays per second. The source is a parallel ray set generated without a random number generator.

Figure 5.5 shows the performance for sequential and parallel ray tracing. The simulation with two threads has a performance of about $6.8 \cdot 10^6$ rays per second. This is around 5% less of twice the sequential performance which is about $3.5 \cdot 10^6$ rays per second. The scheduling attribute of OpenMP has been varied between dynamic and static in a range from 10 and 100000, but no substantial performance difference was detected.

6 Summary and Outlook

This section gives a summary about the results of the different problems that were analyzed.

In Section 2.1, a parallel, a diffuse, and a power cosine ray set are introduced and two test cases to validate their distribution are described. The results of these are discussed in Section 4.1 and show that the implemented ray sets follow the theoretical behavior, except for an expected noise, introduced by the MC sampling.

This noise is analyzed in Section 4.3. The effects of a changing number of rays and a change in the spatial resolution on the noise is evaluated for a rotationally symmetric, cylinder geometry. A higher number of rays decreases the noise whereas a higher number of triangles increases the fluctuations. For a cylinder geometry with about 260k triangles about 10^9 rays are necessary to lower the maximum local error to under 15%.

In Section 2.2, it is described how to use instancing to model the reflective and periodic boundary condition. In Section 2.3, the calculation of the direct flux is explained. The implementation of the direct flux simulation algorithm with Embree is described in Section 3.1. The result of a direct flux rate simulation for a tower geometry is presented in Section 4.2. Effects of instancing and reflective and boundary conditions on the direct flux rate are shown.

In Section 5, results for the performance of the intersection loop are presented for different problems. The performance of the ray intersection loop, which is the most time consuming part of the simulation, is determined mainly by two parts: a) the ray generation operation which depends on the type of the ray source and its implementation and b) the ray tracing algorithm which depends on the BVH structure and the direction and position of the rays. The BVH structure built with Embree depends on the geometry investigated, the number of triangles and the number of instances.

In Section 5.1, results for the performance depending on the ray set and the number of triangles are presented. A performance of about $1.5 \cdot 10^6$ rays per second for the whole loop and $2 \cdot 10^6$ rays per second for the loop without ray generation were measured. A cylinder geometry with about 260k triangles without instancing was investigated.

Section 5.2 provides results for the performance depending on the number of instances. Redundant instances that are not hit do not effect the executionspeed of the loop. It was observed that the size of the ray source domain does effect the intersecting loop substantially. In Section 5.3, results for parallelizing the intersection loop with OpenMP are shown. An almost perfect speed up of about 1,95 was achieved with two cores.

Future work will focus on integrating this boundary condition instancing method into a simulator for process TCAD and compare the results and performance to the classical approach to recalculate the rays at the simulation boundary. Further, optimization possibilities provided by Embree will be implemented to improve the performance of the simulation. The general direction will be to improve the performance of the etch simulation, while at least keeping the MC sampling noise in a given range to provide a certain statistical accuracy.

Bibliography

[1] C.K. Sarkar, *Technology Computer Aided Design: Simulation for VLSI MOSFET* (CRC Press, 2013)

[2] S. Li, Y. Fu, *3D TCAD Simulation for Semiconductor Processes, Devices and Optoelectronics* (Springer Science & Business Media, 2011)

[3] S. Selberherr, *Analysis and Simulation of Semiconductor Devices* (Springer, 1984)

[4] P. Williams, *Plasma Processing of Semiconductors* (Springer Science & Business Media, 1997)

[5] M.A. Lieberman, A.J. Lichtenberg, *Principles of Plasma Discharges and Materials Processing* (John Wiley & Sons, 2005)

[6] A. Fridman, *Plasma Chemistry* (Cambridge University Press, 2008)

[7] O. Ertl, S. Selberherr, Three-Dimensional Plasma Etching Simulation Using Advanced Ray Tracing and Level Set Techniques, ECS Transactions **23**(1), 61 (2009). DOI 10.1149/1.3183702

[8] O. Ertl, S. Selberherr, Three-Dimensional Level Set Based Bosch Process Simulations Using Ray Tracing for Flux Calculation, Microelectronic Engineering **87**(1), 20 (2010). DOI 10.1016/j.mee.2009.05.011

[9] D. Kunder, E. Bär, Comparison of Different Methods for Simulating the Effect of Specular Ion Reflection on Microtrenching During Dry Etching of Polysilicon, Microelectronic Engineering **85**(5–6), 992 (2008). DOI 10.1016/j.mee.2008.01.038

[10] T. Brochu, R. Bridson, Robust Topological Operations for Dynamic Explicit Surfaces, SIAM Journal on Scientific Computing **31**(4), 2472 (2009). DOI 10.1137/080737617

[11] "EMBREE." [Online]. Available: `http://embree.github.io/`

[12] O. Ertl, S. Selberherr, Three-Dimensional Topography Simulation Using Advanced Level Set and Ray Tracing Methods, in *Proceedings of the International Conference on Simulation of Semiconductor Processes and Devices (SISPAD)* (2008), pp. 325–328. DOI 10.1109/SISPAD.2008.4648303

[13] G. Marsaglia, Choosing a Point From the Surface of a Sphere, The Annals of Mathematical Statistics **43**(2), 645 (1972). DOI 10.1214/aoms/1177692644

[14] O. Ertl, Numerical Methods for Topography Simulation. Ph.D. Dissertation, Technische Universität Wien (2010)

[15] P. Shirley, M. Ashikhmin, S. Marschner, *Fundamentals of Computer Graphics* (CRC Press, 2015)

[16] R. Taschner, *Anwendungsorientierte Mathematik für ingenieurwissenschaftliche Fachrichtungen*, vol. 2 (Carl Hanser Verlag, 2014)

[17] "NVIDIA OptiX Ray Tracing Engine." [Online]. Available: `https://developer.nvidia.com/optix`

[18] "Visualization Toolkit (VTK)." [Online]. Available: `http://www.vtk.org/`

[19] J. Jeffers, J. Reinders, *High Performance Parallelism Pearls Volume Two: Multicore and Many-Core Programming Approaches* (Morgan Kaufmann, 2015)

[20] I. Wald, S. Woop, C. Benthin, G.S. Johnson, M. Ernst, Embree: A Kernel Framework for Efficient CPU Ray Tracing, ACM Transactions on Graphics **33**(4), 143 (2014). DOI 10.1145/2601097.2601199

[21] W.J. Schroeder, B. Lorensen, K. Martin, *The Visualization Toolkit* (Kitware, 2004)

[22] W.J. Schroeder, L.S. Avila, W. Hoffman, Visualizing with VTK: A Tutorial, IEEE Computer Graphics and Applications **20**(5), 20 (2000). DOI 10.1109/38.865875

[23] "ParaView." [Online]. Available: `http://www.paraview.org/`

[24] U. Ayachit, *The ParaView Guide: A Parallel Visualization Application* (Kitware, Inc., USA, 2015)

[25] J. Ahrens, B. Geveci, C. Law, ParaView: An End-User Tool for Large-Data Visualization, Visualization Handbook, Elsevier (2005). DOI 10.1016/B978-012387582-2/50038-1

Hiermit erkläre ich, dass die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet.

Die Arbeit wurde bisher weder im In– noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, 22.12.2016

_____
Dominik Koukola