

Geant4 based library "SCoRe4" for Surface Contamination and Roughness Effects simulations in rare event search experiments

Christoph Grüner^{1,2*}

¹Atominstitut, Technische Universität Wien, Stadionallee 2, Vienna, 1020, Austria.

²Marietta Blau Institute, Österreichische Akademie der Wissenschaften, Vienna, 1010, Austria.

Corresponding author(s). E-mail(s): christoph.gruener@oeaw.ac.at;

Abstract

Surface simulations are important for accurately modeling particle interactions in experiments where background contributions from surface contaminants can significantly affect detector performance. In rare event searches, such as dark matter or neutrinoless double beta decay experiments, standard Geant4 simulations typically assume perfectly smooth surfaces, neglecting the microscopic roughness that exists in real materials. This simplification can lead to inaccurate predictions of energy deposition. To address this limitation, I developed SCoRe4, a Geant4-based library designed to simulate more realistic surface roughness based on experimentally measurable parameters. The code allows users to generate patches of simplified rough surface geometries across a wide range of scales — from square millimeters to square meters — while maintaining computational efficiency. SCoRe4 is open source and can be easily integrated into existing Geant4 setups. This work presents the structure, implementation, and example application of SCoRe4, as well as its potential use in improving the accuracy of background modeling in rare event physics.

Keywords: Surface Roughness Simulation, Surface Contamination, Rare Event Search, Dark Matter, Neutrinoless Double Beta Decay, Geant4

1 Introduction

Geant4 (GEometry ANd Tracking) is a widely used toolkit for simulating the propagation and interaction of particles, atoms, etc. with various geometries and materials [1–3] based on Monte Carlo methods. In the field of rare event research, such as in dark matter searches or neutrinoless double beta decay experiments, it is common to construct electromagnetic background models to better understand detector performance and background [4, 5]. Therefore, the geometries and materials of the experimental setup are implemented in Geant4 and various sources that contribute to the

background spectrum are added to the simulation. This often includes atmospheric muons, ambient γ -rays, long-lived radiogenic impurities that are activated by cosmic rays. This also involves bulk and surface contaminants such as ^{210}Pb or ^{238}U , found in the materials used for the experimental setup. Surface contamination is of particular interest, since unlike bulk contamination—where nuclear decays produce a well-defined energy signature—the signal from surface contamination can be distorted or suppressed, as decay products may escape the detector, depositing only a fraction of their energy.

Geant4 offers a range of tools, settings and geometries to model a realistic setup. The user can utilize different volumes like boxes, tubes, polyhedra, etc. combine or subtract them from one another to assemble a representation of the real-world geometry. This results in complex geometric structures. However, the surfaces of the geometries used are flat at the microscopic level. In contrast, actual surfaces exhibit a degree of roughness that is not accounted for in these simulations. Due to this mismatch with reality, the effects of the surface structure can not be accounted for in the simulation. This paper introduces the Geant4-based library SCoRe4, designed to simulate surface roughness based on measured parameters and applicable to virtually any dimension of surface area from a few mm² to m², while maintaining computational efficiency.

This library is open source and published under the GPL-3.0 license, hosted on github [13] and major versions are published on Zenodo [12]. SCoRe4 depends on several publicly available libraries and toolkits: For simulations *Geant4* (version 10.6.3) [1–3]; for calculations *Numpy*, *SciPy* and *Numba* [14, 16, 19]; for data processing and representation *Matplotlib*, *Pandas*, *PyYaml*, *lxml* and *trimesh* [6, 8, 15, 17, 20]; and for plotting of progress bars *tqdm* [9].

This work is based on SCoRe4 release version 1.0.0 [12] and is structured as follows: An initial application of SCoRe4 is discussed in section 2. Multiple library code modules have been developed to address all aspects of simulating a rough surface, including the generation of rough surface geometry, scaling the geometry to represent nearly any surface size, and sampling vertices on and beneath the surface for contamination simulations. These library modules are elaborated in section 3. In conclusion, section 4 provides a future perspective. An example of applying the library can be found in Appendix A.

2 Background Model

Background models in rare event search experiments are important to learn more about all sources contributing to the energy deposition spectrum and to either reduce their contribution or distinguish a genuine signal from them. The development of a comprehensive and detailed

background model that includes all important origins that contribute to the measured spectrum is a tedious task that involves multiple steps. Especially detailed contributions from surface contaminations are hard to model based on the measured surface.

Here, the developed library SCoRe4 adds the important detail of surface contributions to the spectrum. The library allows for detailed simulation of particles that interact with a complex surface structure defined as a geometry in Geant4. This typically computer resource-demanding geometric representation of the surface involving millions of objects is feasible due to advancements within the library.

Incorporating these surface simulations into a background model, such as that developed by the Cryogenic Rare Event Search with Superconducting Thermometers (CRESST) experiment [5], requires integrating SCoRe4 into the experimental background model development workflow. For the CRESST experiment, SCoRe4 was used to add surface roughness to the recreation of the experimental configuration within the Geant4 framework. To create a background model the workflow includes the implementation of detectors, shielding, and various other components that have contaminants or that might contribute in other ways to the measured background. Suitable Geant4 physics lists must be set for different interactions or physics such as the Coulomb interaction or decay physics. The simulation domain incorporates multiple internal and external particle sources. The aggregate energy deposited by each particle in the detector forms energy deposition templates for each particle type and source. Subsequently, these templates are further processed to reflect the detector detection efficiency and then fitted to the experimental data.

The new library presented in this paper adds for the first time surface roughness to the experimental configuration within the Geant4 framework and allows for sources representing a contamination of these surfaces. This opens up the possibility of accounting for more effects in the simulation, such as the surface roughness of the target, but also for the surrounding material, which is important for understanding the measured background in detail.

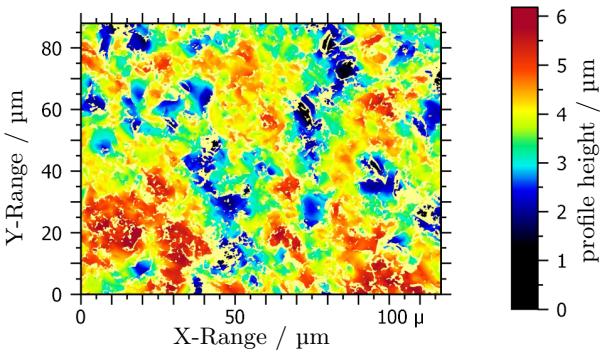


Fig. 2: Surface profile of a diffused crystal surface. Structures in the μm scale are visible. Figure is taken from [11]

3 Library

The surfaces of crystals used as target materials in rare event search experiment can be polished and therefore flat without nearly any structure or diffused with complex structures in the μm range as can be seen in Figure 2. Simulating a diffused contaminated surface of small scale (μm) structure can be tricky as even for a simplified representation of these structures by simple spikes, the number of needed spikes increases quadratically with surface area. Having a surface area of only 1 cm^2 and a spike base of the size of $100\text{ }\mu\text{m}^2$ already results in 10^6 spikes to cover this area. Covering slightly bigger areas or using spikes with a smaller base quickly exceeds the computers working memory.

Further, to simulate surface contamination the points to place the contaminants on and below the surface following some distribution depending on the depth below the surface must be defined. Default Geant4 provides different particle generators to define such points, however, sampling on uneven surfaces cannot be done out of the box. To address these problems, three different modules are implemented in the **SCoRe4**: **a) SurfaceRoughness**, **b) Portal**, and **c) ParticleGenerator**. All modules combined allow one to perform detailed surface roughness contamination simulations.

3.1 Module: Surface Roughness

This module generates a rough surface patch represented by multiple spike-shaped structures (Figure 3) ordered in a grid like manner (Figure 4). A single structure can be either one physical volume or a combination of multiple physical volumes. This allows for different-looking spike-like structures and a more complex surface. However, all spikes in a batch still look alike. The dimension of the basis of this structure must be the same for all spikes in a single patch of rough surface.

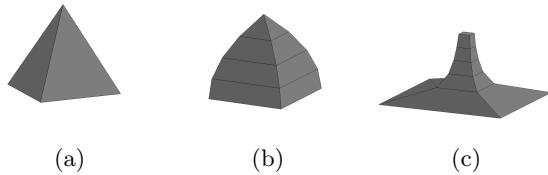


Fig. 3: Visualization of different shape implementations for spikes, which can be of any size that is allowed in Geant4.: (a) simplest form of a spike, basis is a Geant4 tetrahedron. (b) multiple layers of Geant4 tetrahedrons form the spike, the outer surface approximates a squared function. (c) multiple layers of Geant4 tetrahedrons form the spike, the outer surface approximates $1/x$.

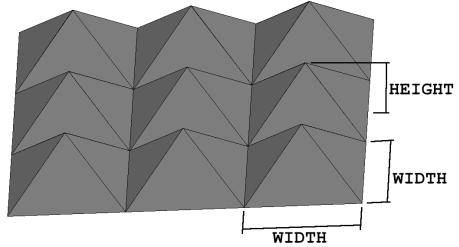


Fig. 4: Visualization of 3×3 spikes placed at the surface of a target volume to simulate a patch of rough surface. The generated spikes can be of any size allowed in Geant4. The size is controlled via setting a width and height for the spiked.

A surface patch is represented by a G4MultiUnion and can contain up to $\sim 1000 \times 1000$ spikes. This number is limited by Random-Access Memory (RAM) which arises due to the voxelization process for G4MultiUnion

(reference to voxelization). It is an optimization algorithm that divides the simulation domain of G4MultiUnion into small boxes (voxels) to improve the tracking of a simulated particle within the domain. The number of boxes and, thus, the memory consumption increases with $\mathcal{O}(n^3)$ if the volume boundaries are not exactly on the same axis. The maximum number of maximal splits in one dimension is hardcoded in Geant4 and set to 10^5 and can result in 10^{15} boxes, which exceed the RAM of a standard computer. To change this number, the voxelization class was adapted.

The module also includes a calculator to compute the surface parameters [18]: Arithmetical Mean Height (Sa), Root Mean Square Height (Sq), Skewness (Ssk) and Kurtosis (Sku).

$$\begin{aligned} Sa &= \frac{1}{A} \iint_A |z(x, y)| dA \\ Sq &= \sqrt{\frac{1}{A} \iint_A z^2(x, y) dA} \\ Ssk &= \frac{1}{Sq^3} \left[\frac{1}{A} \iint_A z^3(x, y) dA \right] \\ Sku &= \frac{1}{Sq^4} \left[\frac{1}{A} \iint_A z^4(x, y) dA \right] \end{aligned}$$

The surface is described by a small bundle of parameters and must be set by the user: **Spike-form** sets the general shape of the simulated spikes (see Figure 3). **Width X/Y** describes the width of a single spike basis in the X and Y directions. **Nr X/Y** describes the number of spikes in the X and Y directions. **MeanHeight** sets the mean height of the spikes. **Deviation** defines the one- σ deviation of the height of the spikes from their mean height. If set to zero, the mean height is the final height. Some spike shapes require multiple layers to be formed, and the number of layers can be set with **Nr layer**. All values can be either set via code during the generation of the detector or using a macro-file.

The whole generation of the patch of rough surface is controlled by the *Generator* class. This class handles the *Describer* and *Assembler* class (see Figure 5). To generate *G4MultiUnion*, the aforementioned parameters are passed on and interpreted by the *Describer* class. The class creates a list of all needed *G4Solids* inclusive of their parameters (height, width, etc.) and relative position in the rough surface patch. The *Assembler* then creates the *G4MultiUnion* and adds all listed solids. In parallel, the *FacetStore* is filled with

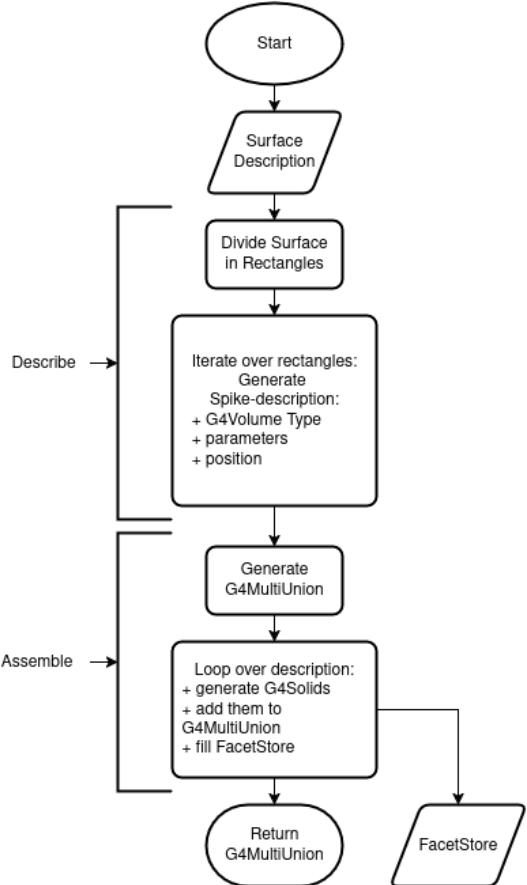


Fig. 5: The flowchart shows the generation of a patch of rough surface, starting with passing the user defined surface description to the class. Next the chart is divided in two parts: 1) *Describe* represents the actions of the class *Describer* which translates the user description into a list of needed *G4Solids* inclusive their parameters and location. This list is passed to the *Assembler* class which generates all the needed solids and adds them to a *G4MultiUnion*. In parallel the *FacetStore* is filled.

the correct facets that represent the exact surface. Finally, the generator returns a handle to a *G4MultiUnion*.

3.2 Module: Portal

For the simulation of a diffuse / roughened surface, the surface roughness module subsection 3.1 simulates a patch of single spikes in the μm scale each. To simulate at least a rough surface area of 1 cm^2 this results in at least a million spikes.

Simulating bigger surface areas therefore exceeds computational resources such as random access memory, limiting the simulation domain to an order of mm. In the following section, we introduce the portal-subworld structure, in which the portal volume is represented by a small number of subworlds that are reused to model the entire portal region (see [Figure 6](#)). Placing the complex, computationally intensive geometry inside such a subworld allows the user to reuse it multiple times, without the high increase in needed resources. Hereby, the portal volume acts as a placeholder and maps the covered volume to the corresponding single or multiple different subworlds. For that, the portal volume contains a grid-like map that divides the volume into equally sized domains and assigns a subworld to it (see [Figure 7](#)). A subworld can be assigned multiple times. When a particle enters the portal volume, it is moved to the corresponding subworld on the basis of its entry position. Inside the subworld the particle can interact with the geometries inside. Upon leaving a subworld, the particle is either moved to the next corresponding subworld, re-enters the same subworld, or exits the portal volume at the corresponding position. This is based on the position of the particle in the portal grid. When a particle "moves" through the portal volume, it actually moves through the subworlds volume, but also does integer steps on the grid map when passing the subworlds boundaries. This approach allows for the simulation of large and complex domains while requiring only a modest amount of memory.

The portal mechanism is controlled by the *G4UserSteppingAction* class. Each simulated particle step the mechanism checks if the particle must be moved. This happens in two cases: the particle enters the portal or the trigger volume. The trigger volume is part of the subworld and surrounds the subworlds simulation domain. It guarantees for a fast and easy check that the particle left the subworld.

Unlike the portal volume, which is deliberately positioned within the simulation domain to enable particle interactions, the subworld must be located beyond the range of particles in the domain, since particles are intended to enter it exclusively through the movement mechanism governed by the *G4UserSteppingAction*.

As setting up the portal and subworlds can be complex, portal-, subworld-, trigger-volumes

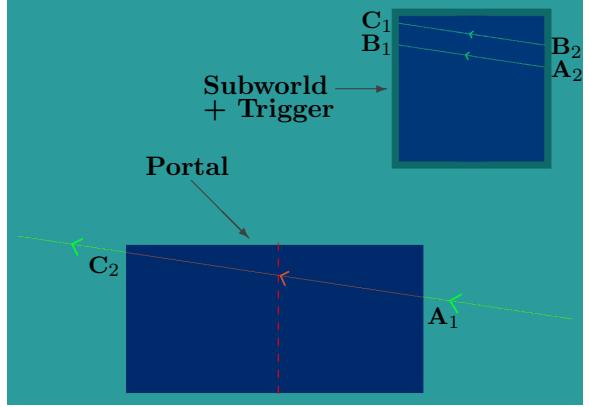


Fig. 6: 2D sketch of the function of the implemented Portal representing two subworlds. The green dotted line is the trajectory of the particles entering the portal and the subworld on the right side and leaving them on the left. The orange dotted line is the imagined trajectory of the particle crossing the portal. The red dashed line divides the portal into two subworlds.

The simulated particle undergoes the following steps: A) it enters the portal on the right side at A_1 and is ported to point A_2 of the subworld without a change in momentum. B) after crossing the subworld it leaves the volume at B_1 and enters the trigger. This activates the periodic teleportation and the particle is set to point B_2 without a change in momentum. C) after crossing the subworld for the second time it leaves the subworld's volume at C_1 and enters the trigger again. The particle is teleported to C_2 where it leaves the portal, without a change in momentum. The dimension of the subworlds and the portal can be of any size allowed by Geant4. The trigger volume ensures a correct activation of particle teleportation.

must be created; a grid linking the portal and the subworlds must be filled; and the subworld-simulation domain placed inside its trigger. To support the user, a helper class is provided. The following parameters must be set: **Dx/Dy/Dz-Sub** is the size of the subworld in x-, y- and z-direction, **Dx/Dy/Dz-Portal** is the size of the portal in x-, y- and z-direction, **Nx/Ny/Sub** sets the grid-size in x- and y-direction, **nSubwords** sets the number of different subworlds, **Portal-Name** and **SubworldName** define the portals and subworlds name, with **MotherVolume** the mother volume is set, **PortalPlacement** defines

A	A	C	B	A	C	B	A
A	B	A	B	B	A	B	A
C	A	A	B	A	B	A	B
B	B	A	A	A	C	A	B
A	C	B	B	A	B	B	A
B	A	B	C	B	A	B	B

Fig. 7: Subworlds can be combined in a grid to represent a more divers and bigger subworld. Thereby, already a small number of subworlds can be used to simulate a bigger domain. The grid can be filled manually or randomly with a user defined distribution. In this visualization three subworlds A, B and C are assembled in a 8×6 grid with set probabilities for A: 44%, B: 44%, C: 12%. Because of the finite number of grid points, the final distribution of subworlds differs from the defined one.

the global position of the portal, **SubworldPlacement** is a vector of all global subworld positions, **SubworldDensity** sets the density of the subworlds in the grid and **SubworldMaterial** defines the material of the subworlds simulation domain.

3.3 Module: Particle Generator/Shift

With a portal-subworld setup and a rough surface in place, the sampling of the starting vertices with respect to the surface structure cannot be performed by Geant4's default particle generator. Therefore, this library provides a custom particle generator that samples vertices uniformly distributed over the rough surface structure. This works even in the case of a portal-subworld setup with different surface structures in different subworlds.

The uniform sampling is based on the facet store, introduced in subsection 3.1 and the portal grid from subsection 3.2. The facet store holds a representation of the exact outer surface of the surface roughness in the form of *G4TriangularFacets*. The sampling of uniformly

distributed points is based on the surface area and executed in three steps: 1) uniform sampling of a subworld in the portal grid based on the total surface area in the corresponding subworld; 2) sampling of *G4TriangularFacet* in the corresponding facet store based on the facets area; 3) sampling of point on the facet using Geant4s sampling mechanism.

The particle generator samples points exactly at the surface, however, to start particle simulations in the vicinity below the surface, the sampled vertices have to be shifted. This is done by the shift module, which takes a sampled vertex and moves it in a certain direction. The direction and size of the shift can be defined by the user. However, in the case of surface contamination simulation, the direction is perpendicular to the surface at which the vertex is sampled, pointing below the surface.

A sampling function can be passed to the module in the form of a histogram-like list of step in nm and number of expected vertices at this step. In addition, sampling can be restricted to an area by setting a minimal and maximal shift or by restricting to a material. The sampler linearly interpolates the missing values in between. In Figure 8 the actual sampling can be seen in comparison to the set values including a minimal (10 nm) and maximal (110 nm) shift and restriction to the volumes material.

3.4 All modules combined

All single modules complete a certain task, such as reusing a small simulation domain again to represent a larger one, creating a rough surface structure, or sampling and shifting staring vertices for simulation. To perform a full surface contamination simulation of a rough surface, all of these modules must be linked together. Extending the rough surface patch is rather simple as it must only be placed within the reused subworld. In addition, when generating the rough surface patch, the module also creates the surface store. This must be placed in the position of the rough surface patch. An extra positioning is necessary, as the generated *G4MultiUnion* representing the rough surface patch is a *G4Solid* and does not have a final position. It must be placed as a physical volume in the simulation domain. Further, the facet store must be linked to the subworld just by

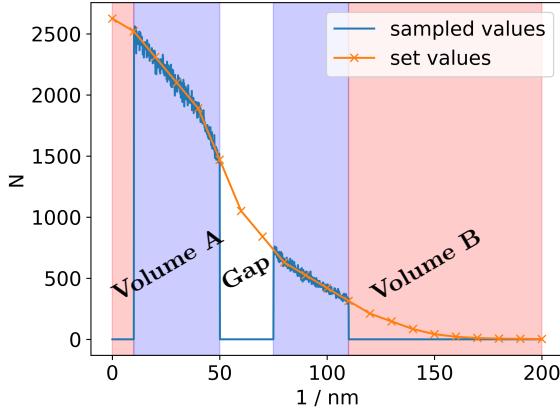


Fig. 8: Example of sampled values for a shift (blue) based on a defined distribution (orange, the x marker are the set values in the distribution file, the line represents the linear interpolation between them) in the range of nm. In the example, two Geant4 volumes (marked as red and blue areas) are placed in line with a gap (50 to 75 nm) between them. Although the distribution is non zero in the areas marked as red (0 to 10 nm, 110 to 200 nm) and white (50 to 75 nm) no points are sampled in this ranges due to special options set for shifting: a) a minimal shift of 10 nm is set, therefore no values are sampled in the first red marked area. b) a maximal shift of 110 nm is set, therefore no values are sampled in the second red marked area. c) the shift is confined to the volumes material but the white area is a gap between the two volumes, therefore not points are sampled from (50 to 75 nm).

passing the pointer of the facet store. Finally, the name of the portal must be added to the sampler. For the simplification of the setup process, helper classes are provided. (For an example setup, see [Appendix A](#).)

3.5 Extension Module: Complex Surface Generation

The introduced modules enable the simulation of a simplified patch of a rough surface over a large area using spikes. However, the portal module is versatile and can be used with any geometry inside. Therefore, more complex surfaces can be generated and utilized. The additional Python tool *SurfaceGenerator.py*, which is a part of SCoRe4, was developed for this purpose. Using

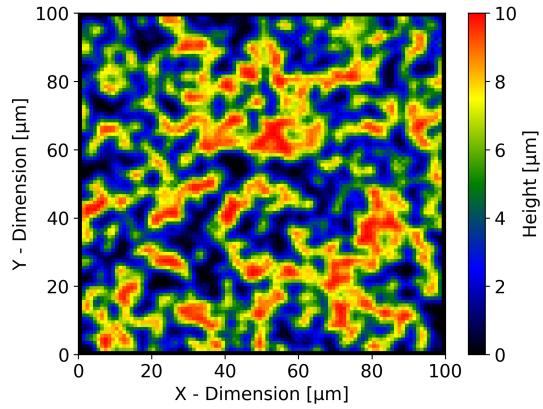


Fig. 9: Example of surface profile generated with SCoRe4. Structures in the μm scale are visible.

this tool, surface profiles (see [Figure 9](#)) can be generated and exported to a *GDML* file [7]. The tool provides different routines to generate more complex surfaces and can be used via the command line. In this file, the surface is stored as a *G4Tessellated* object, which can be loaded into Geant4 using the class *LogicalSurface*. Although *GDML* files can be loaded without the provided class, its usage is highly recommended, as the class is also used to sample evenly distributed points from the surface later in the Geant4 simulation process.

However, in comparison to the simpler surface representation using spikes and a portal, the use of *G4Tessellated* objects increases computation time and is currently about two magnitudes slower. An example of an application is shipped with the library.

4 Conclusion

This paper presents the developed Geant4-based C++ library SCoRe4 to add surface roughness in combination with surface contamination to simulations. This allows, for the first time, the simulation of surface roughness based on real measured surface parameters. This is especially important for rare event research experiments such as CRESST [5] or CUORE [4], as the effects of surface contamination have an impact on the measured electromagnetic background spectrum. Therefore, a detailed background simulation is of the utmost importance. The library was developed

and tested on the basis of the CRESST experiment [10, 11], and I was further able to reproduce some results of the CUORE experiment. For more information, see [11].

The library SCoRe4 comes with three C++ code modules: to simulate a small patch of rough surface; extend the patch of rough surface to a large surface area of your choice; and a particle generator to sample starting vertices on the patch of rough surface and shift them below the surface. All these modules combined now allow for simulations of rough and contaminated surfaces.

A simulation result is provided in Figure 10 where α -particles with an energy of 5.3 MeV are distributed uniformly on a flat or rough $3 \times 3 \text{ cm}^2$ surface of a silicon crystal. This energy corresponds to the typical energy of α -particles emitted during the decay of Po^{210} in the Pb^{210} decay chain, a potential surface contaminant [11]. The rough surface is characterized by spikes measuring $10 \mu\text{m}$ in both height and width. The effect of a roughened surface is clearly visible for energies below 4 MeV by a higher number of counts.

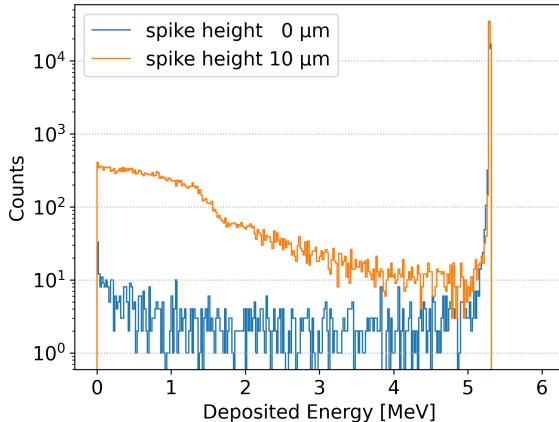


Fig. 10: Energy deposition spectrum of α -particles with an energy of 5.3 MeV placed at a $3 \times 3 \text{ cm}^2$ surface without and with spikes of height and width of $10 \mu\text{m}$.

An additional Python code module provided with a command line interface can be used to generate a more complex surface and export it to a *GDML* file. This is supported by further Geant4-based C++ classes to correctly load the surface volume from the *GDML* file, place it correctly in simulation and get access to uniform point

sampling from its surface. Despite offering more detailed surface simulations, this newly developed module demands significant computational resources, resulting in extended runtimes.

The open source library SCoRe4 provides a foundation for surface roughness and contamination simulations in Geant4 and should pave the way toward a better understanding of the effects of microscopic surface structures in rare event research experiments.

Following this publication, the library will be maintained and further extended to improve simulation speed using *G4Tessellated* surface objects, as well as to provide additional options for reconstructing more realistic surfaces.

Code examples and additional usage instructions are shipped with the library.

Interested users are encouraged to participate in testing, giving feedback, reporting bugs, or contributing to the continued development of the library [13].

Acknowledgements. I would like to thank the CRESST collaboration for the many discussions and valuable input. In particular, I would like to acknowledge the simulation group of CRESST — Valentyna Mokina, Samir Banik, Holger Kluck, Jens Burkhart and Robert Breier — for their contributions. I am especially grateful to Valentyna Mokina for her continuous feedback on the progress of this paper.

This work has been funded in part by the Austrian Science Fund (FWF) by <http://dx.doi.org/10.55776/I5420>.

A Example

For setting up a simulation using SCoRe4 multiple steps must be completed. In the following section, an example is presented and important details are discussed. The goal is to establish a rough surface patch of silicon, an area of $10 \text{ cm} \times 10 \text{ cm}$ and a roughness in the range of μm . In a next step, we introduce ^{210}Po as contamination on the surface. Setting up this simulation involves: the preparation and linking of a portal and subworld; the activation of the teleportation mechanism; the definition of a rough surface and its placement in the subworld; and the setup of the particle generator. The most important details of each step are discussed. For a description of the named modules,

see section [section 3](#). More examples and further explanation can be found with the code [\[12, 13\]](#).

A.1 Setup of the Simulation

In the following, the most important steps in preparing the simulation are presented. It is assumed that the user has some familiarity with the Geant4 simulation framework, particularly with the *G4VUser...* classes, such as *G4VUserDetectorConstruction* or *G4VUserPrimaryGeneratorAction*.

A.1.1 Portal - Subworld

The key element for a surface simulation with macroscopic areas is the portal-subworld module. To simplify its setup, a helper class *Surface::MultiportalHelper* is provided, which first gathers all the needed information, such as the size of the portal, size of the subworld, number of different subworlds and their frequency, materials, etc. After providing the information, the helper generates a portal-subworld pair ([Listing 1](#)).

To activate the portal mechanism, the control class must be added to the *G4UserSteppingAction* and activated by calling the *DoStep(...)* function. ([Listing 2](#)).

Listing 1: Setup of Portal-Subworld pair. Class: G4VUserDetectorConstruction

```
#include "MultipleSubworld.hh"
#include "MultiportalHelper.hh"

G4VPhysicalVolume
    ↪ *DetectorConstruction::Construct() {
    ...

    Surface::MultiportalHelper helper("Helper",
        ↪ 5);

    //define portal
    helper.SetPortalName("Portal");
    helper.SetDxPortal(10 * cm);
    helper.SetDyPortal(10 * cm);
    helper.SetDzPortal(0.5 * cm);
    helper.SetPortalPlacement(trafоОPortal);

    //set size of subworld
    helper.SetDxSub(0.5 * mm);
    helper.SetDySub(0.5 * mm);
    helper.SetDzSub(0.5 * cm);

    //set number of subworlds (must match with
    ↪ subworld- and portal size)
    helper.SetNxSub(200);
    helper.SetNySub(200);

    //set number of different subworlds and
    ↪ material
    helper.SetNDifferentSubworlds(3);
    helper.SetSubworldMaterial(subworldMaterial);
```

```
//place subworlds
helper.AddSubworldPlacement(trafоА);
helper.AddSubworldPlacement(trafоБ);
helper.AddSubworldPlacement(trafоС);

//set density of subworlds
helper.AddSubworldDensity(0.3);
helper.AddSubworldDensity(0.5);
helper.AddSubworldDensity(0.2);

//set mother volume for portal-subworld setup
helper.SetMotherVolume(logicWorld);

//generate setup
helper.Generate();
...
return physWorld;
}
```

Listing 2: Activation of Portal-Subworld, File: SteppingAction.cc

```
#include "PortalControl.hh"

void SteppingAction::UserSteppingAction(const
    ↪ G4Step *step){
    fPortalControl.DoStep(step);
}
```

A.1.2 Surface Structure

The generation of a rough surface can be accomplished by using the *Surface::RoughnessHelper* class ([Listing 3](#)). Based on a description (number of spikes, their size and shape, etc.) that can be done in code or via a macro file ([Listing 4](#)), this class returns a *G4LogicalVolume* that represents a rough surface patch and must be handled accordingly.

Listing 3: Generation of surface based on macro file

```
G4VPhysicalVolume
    ↪ *DetectorConstruction::Construct() {
    ...
f_surface_helper.Generate();
G4LogicalVolume *logical_roughness =
    ↪ f_surface_helper.LogicRoughness();
...
}
```

In our setup, we introduce macro files similar to those used in Geant4. These macro files act as control scripts for key simulation options, such as enabling or adjusting surface roughness generation, or configuring the portal subworld setup. The main benefit is flexibility: Parameters can be changed directly in the macro file without modifying or recompiling the source code, making testing faster and easier.

Listing 4: Macro commands for surface generation

```
/Surface/RoughnessHelper/setVerbose 2

/Surface/RoughnessHelper/setBasisDx 0.5 mm
/Surface/RoughnessHelper/setBasisDy 0.5 mm
/Surface/RoughnessHelper/setBasisDz 0.25 mm

/Surface/RoughnessHelper/setSpikeDx 5 um
/Surface/RoughnessHelper/setSpikeDy 5 um
/Surface/RoughnessHelper/setSpikeMeanHeight 5 um
/Surface/RoughnessHelper/setSpikeDevHeight 1 mm
/Surface/RoughnessHelper/setSpikeform
    ↪ StandardPyramid

/Surface/RoughnessHelper/setSpikesNx 100
/Surface/RoughnessHelper/setSpikesNy 100

/Surface/RoughnessHelper/setMaterial G4_Si

/Surface/RoughnessHelper/setBoundaryNx 1000
/Surface/RoughnessHelper/setBoundaryNy 1000
/Surface/RoughnessHelper/setBoundaryNz 1000
```

A.1.3 Particle Generator

The particle generator can evenly distribute points on the rough surface with respect to its placement in a portal subworld. For that, the generated rough surface must be linked to the portal. This can be done using the *Surface::MultiportalHelper* and *Surface::RoughnessHelper* classes Listing 5. Afterwards, the *Surface::MultiSubworldSampler* class can be used for sampling points in *G4VUserPrimaryGeneratorAction*.

Listing 5: Setup of particle generator

```
G4VPhysicalVolume
    ↪ *DetectorConstruction::Construct() {
...
f_surface_helper.Generate();

G4Transform3D trafo_surface{
    ↪ G4RotationMatrix(), G4ThreeVector(0.,
    ↪ 0., +f_surface_helper.GetBasisHeight()
    ↪ * 2 - f_portal_helper.GetPortalDz())};

f_portal_helper.AddRoughness(
    ↪ f_surface_helper.LogicRoughness(),
    ↪ trafo_surface,
    ↪ f_surface_helper.FacetStore());

f_portal_helper.Generate();
...
}
```

References

- [1] Agostinelli, S., Allison, J., et al.: Geant4—a simulation toolkit. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **506**(3), 250–303 (2003) [https://doi.org/10.1016/S0168-9002\(03\)01368-8](https://doi.org/10.1016/S0168-9002(03)01368-8)
- [2] Allison, J., Amako, K., et al.: Geant4 developments and applications. IEEE Transactions on Nuclear Science **53**(1), 270–278 (2006) <https://doi.org/10.1109/TNS.2006.869826>
- [3] Allison, J., Amako, K., et al.: Recent developments in Geant4. Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment **835**, 186–225 (2016) <https://doi.org/10.1016/j.nima.2016.06.125>
- [4] Alduino, C., Alfonso, K., et al.: The projected background for the CUORE experiment. The European Physical Journal C **77**(8) (2017) <https://doi.org/10.1140/epjc/s10052-017-5080-6>
- [5] Abdelhameed, A.H., Angloher, G., et al.: Geant4-based electromagnetic background model for the CRESST dark matter experiment. The European Physical Journal C **79**(10) (2019) <https://doi.org/10.1140/epjc/s10052-019-7385-0>
- [6] Behnel, S., Faassen, M., Bicking, I.: lxml: XML and HTML with Python. Lxml (2005)
- [7] Chytracek, R., McCormick, J., Pokorski, W., Santin, G.: Geometry description markup language for physics simulation and analysis applications. IEEE Transactions on Nuclear Science **53**(5), 2892–2896 (2006) <https://doi.org/10.1109/TNS.2006.881062>
- [8] Dawson-Haggerty et al.: Trimesh. <https://trimesh.org/>
- [9] Costa-Luis, C., Larroque, S.K., et al.: Tqdm: A Fast, Extensible Progress Bar for Python and CLI. <https://doi.org/10.5281/zenodo.14231923>
- [10] Grüner, C., Angloher, G., et al.: Geant4 simulations of the influence of contamination and roughness of the detector surface on background spectra in CRESST, vol. TAUP2023, p. 092 (2024). <https://doi.org/10.22323/1.4231923.092>

- [11] Grüner, C.: Enhancing the electromagnetic background model in the cresst experiment by considering surface-induced contamination using measured surface roughness. Master's thesis, Technische Universität Wien (2024). <https://doi.org/10.34726/hss.2024.124144>
- [12] Grüner, C.: SCoRe4 - Geant4 Based Library for Surface Contamination and Roughness Effects Simulations V1.0.0. <https://doi.org/10.5281/zenodo.17648567>
- [13] Grüner, C.: SCoRe4 - Github. <https://github.com/Veltly/G4Surface>
- [14] Harris, C.R., Millman, K.J., *et al.*: Array programming with NumPy. *Nature* **585**(7825), 357–362 (2020) <https://doi.org/10.1038/s41586-020-2649-2>
- [15] Hunter, J.D.: Matplotlib: A 2d graphics environment. *Computing in Science & Engineering* **9**(3), 90–95 (2007) <https://doi.org/10.1109/MCSE.2007.55>
- [16] Lam, S.K., Pitrou, A., Seibert, S.: Numba: a llvm-based python jit compiler. In: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC. LLVM '15. Association for Computing Machinery, New York, NY, USA (2015). <https://doi.org/10.1145/2833157.2833162>
- [17] Simonov, K.: PyYAML Module. <https://github.com/yaml/pyyaml>
- [18] Technical Committee: ISO/TC 213: Iso 25178-2 geometrical product specifications (gps) — surface texture: Arealpart 2: Terms, definitions and surface texture parameters. Technical report, ISO (2021)
- [19] Virtanen, P., Gommers, R., *et al.*: SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* **17**, 261–272 (2020) <https://doi.org/10.1038/s41592-019-0686-2>
- [20] McKinney: Data Structures for Statistical Computing in Python. In: Walt, Millman (eds.) Proceedings of the 9th Python in Science Conference, pp. 56–61 (2010). <https://doi.org/10.25080/Majora-92bf1922-00a>