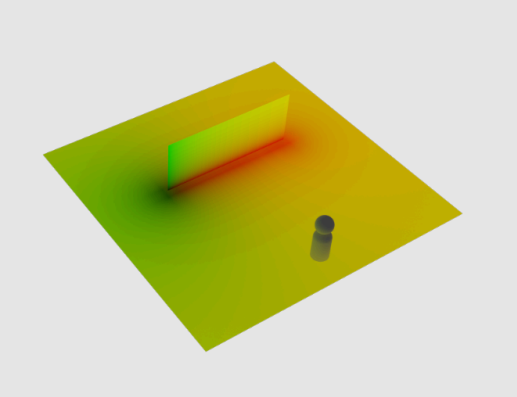
WWDC 2024 Diffuse Reflection UV Computation Tool

***NOTE:** Please see the Enhance the immersion of media viewing in custom environments WWDC video page for the latest information.*

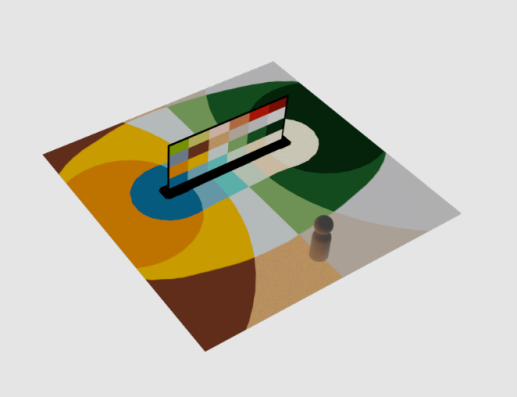
The Diffuse Reflection node in Reality Composer Pro requires two UV inputs: emitter UVs and attenuation UVs. This document explains how to compute them, and provides a python script for automatically computing them within USD files, using a few user-provided parameters.

1. Emitter UVs

Emitter UVs store the average contribution of the docking region UVs, for every vertex in the geometry for which you want to compute diffuse reflection UVs. They are essentially a projection of the docking region UVs onto your input geometry.



Above left: the emitter UVs generated from the docking region, visualized.



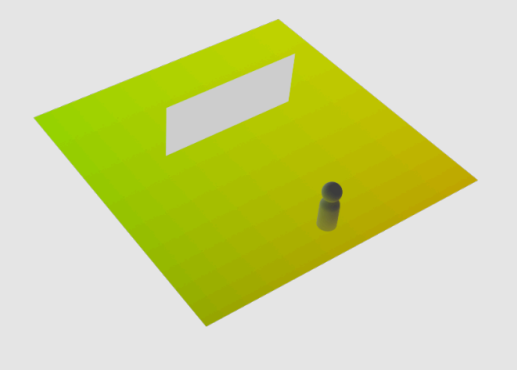
Above right: a debug texture applied to the emitter UVs.

The algorithm which computes emitter UVs iterates over each mesh vertex. For each mesh vertex, it samples a configurable number of random points on the docking region. For each docking region sample, it computes the UV value and weighs it by that point's distance to the mesh vertex, and angle with the mesh vertex. The final emitter UV at that mesh vertex is the average of these weighed docking region UV values.

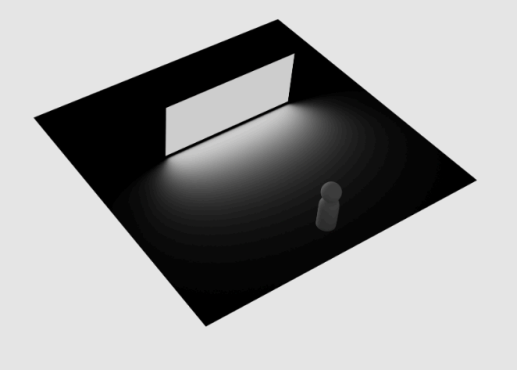
In practice, you don't need to do this manually as our python script handles it entirely. The number of random points sampled from the docking region can have a large impact on the overall computation time, so you can configure a specific number of samples if you wish.

2. Attenuation UVs

Attenuation UVs are a top-down projection of the attenuation texture onto the input geometry. It's helpful to define some common measurement terms so we can reason about the attenuation texture and how it maps on to the input geometry.



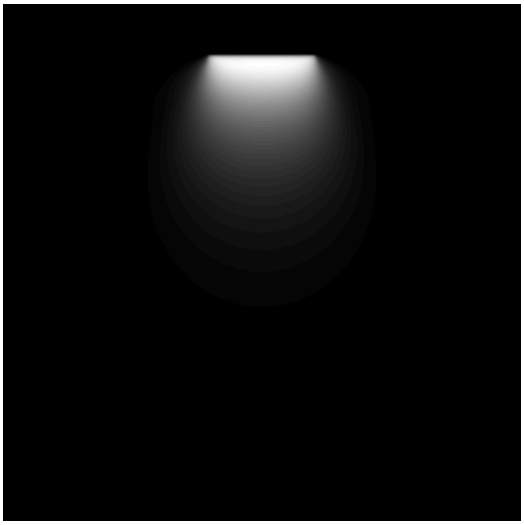
Above left: the attenuation UVs generated from the docking region, visualized.



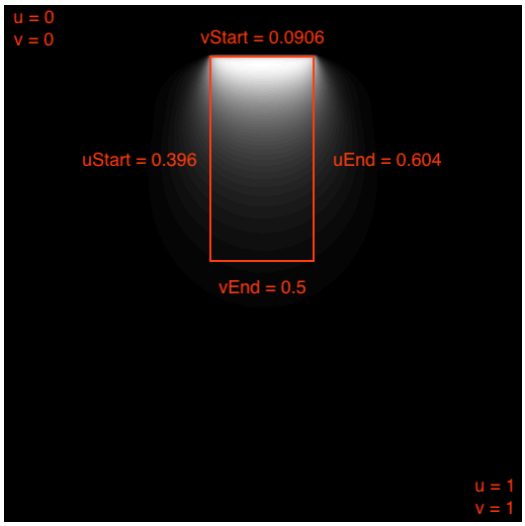
Above right: the default Reality Composer Pro attenuation texture applied to the attenuation UVs.

2A. MEASURING ATTENUATION TEXTURES

The attenuation texture contains a pattern within it which shapes the “falloff” of the diffuse reflection. For reference, the default Reality Composer Pro attenuation texture is below:



Notice how the pattern doesn’t extend all the way to the edges of the texture? Instead, we can think of it as being contained in a rectangle which is smaller than the texture. The rectangle is as wide as the sharp edge of the pattern. It’s the edges of this rectangle we need to measure:



In concrete terms, uStart is the UV-space value (assuming a top-left = `(0,0)` and bottom-right = `(1,1)` UV coordinate system), where the sharp line of the falloff pattern starts horizontally.

uEnd is the UV-space value where the sharp line of the falloff pattern ends horizontally.

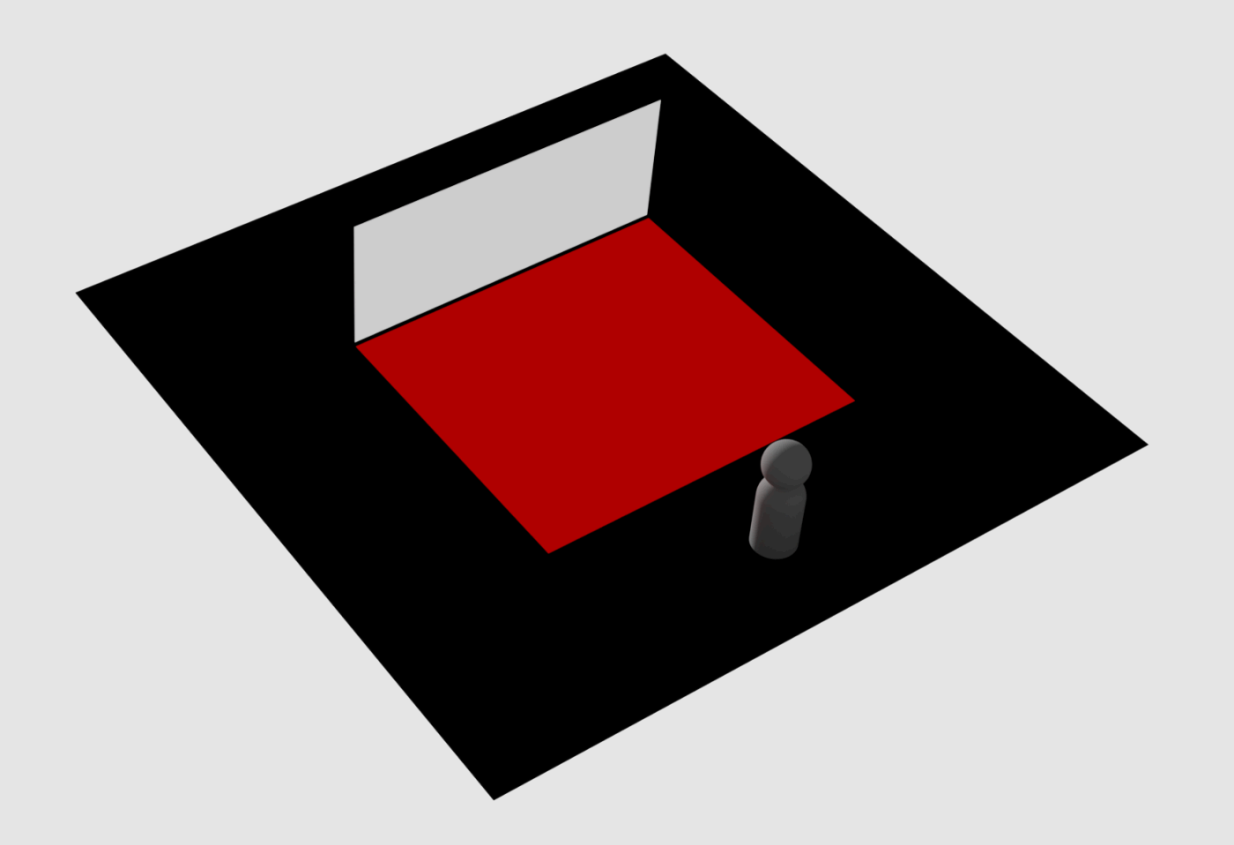
vStart is the UV-space value where the sharp line starts, vertically.

vEnd is the UV-space value where the falloff pattern ends in black.

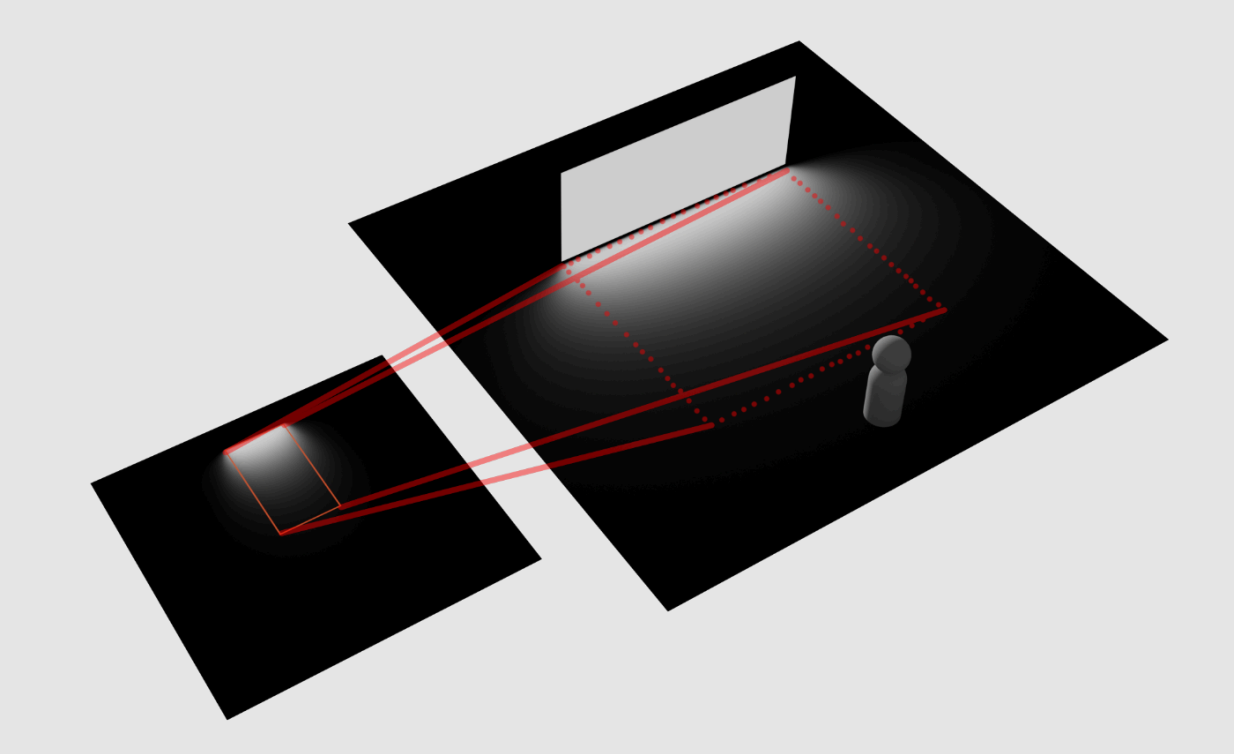
2B. MAPPING ONTO GEOMETRY

Now that we can measure an attenuation texture, we can map it onto our geometry.

In the example below, a square whose sides are equal to the width of the docking region (and which is situated such that the edge aligns with the docking region and extends towards the user at the origin) is highlighted in red.



The attenuation UVs are defined as the mapping from this square, in world space, to the smaller rectangle defined by the uStart/End and vStart/End values, in UV space. In the example below, the highlighted red square has been replaced by the attenuation texture (with the uStart/End/vStart/End rectangle still drawn, for visualization purposes).



This is how the attenuation UVs are constructed: by mapping the world-space vertices from the docking-region-sized-square on the input geometry in front of the docking region, to the smaller rectangle within the attenuation texture.

In practice, you do not have to do this mapping yourself. You just need to measure the uStart, uEnd, vStart, and vEnd values of your attenuation texture, and plug them into the python script provided below. If you are using the default attenuation texture in Reality Composer Pro, then you don't even need to do that, the script will use appropriate default values.

3. Python Tool

We wrote a little python tool which can generate these UVs for you. It requires a python 3 installation, which access to the USD python library. If you do not have the USD python library installed, you can install it with `pip3 install usd-core`.

Usage is as follows:

```
python3 computeDiffuseReflectionUVs.py <input.usd> -o <output.usd> -p / -r true
```

The full arguments list is below:

- `-o` or `--out`: the name of the output file. The output file is an exact copy of the input file, with the two newly generated diffuse reflection UV primvars added.
    - e.g. `-o myNewGeo.usd`
    - This argument is **required**, and ideally should not be the same as the input file.
    - NOTE: the output is essentially a copy of the input USD file, not a reference. If you make changes to the input USD file, you will need to re-run the script to pick up those changes.
- `-p` or `--prim`: the name of the USD prim corresponding to the USD Mesh object(s) you want to compute diffuse

reflection UVs for.
    - e.g. `--prim "/"`
    - This argument is **required.**
    - NOTE: Diffuse reflection UVs are generated for USD Mesh objects only, so if you are using instances in your input file, the generated UVs will not be correct since the UVs are computed per-mesh, not per-instance. Any instanced meshes for which you want to generate diffuse reflection UVs need to be de-instanced into their own unique Mesh objects.
    - NOTE: The algorithm relies on the USD Mesh points. If the points attribute is time-sampled, it will only use the first time-sample. (Time-sampled lightspill UVs are not supported).
- `-r` or `--recursive`: if present, computes diffuse reflection UVs for the USD prim specified by `-p`, and all of its descendants.
    - e.g. `-r true`
    - This argument is optional, the default is “false”.
- `--onlyWithSubstring`: if present (and if `-r` is present), ignores USD prims whose name does not contain the specified substring.
    - e.g. `--onlyWithSubtring "lightspill"`
    - This argument is optional, the default is off.
- `-x` or `--dockingRegionCenterX`: the world-space X component of the center of the docking region.
    - e.g. `-x 0`
    - This argument is optional. The default is 0, and the expected value is 0.
- `-y` or `--dockingRegionCenterY`: the world-space Y component of the center of the docking region.
    - e.g. `-y 4.6`
    - This argument is optional. The default is 0.5, and the expected range of values is -infinity to +infinity.
- `-z` or `--dockingRegionCenterZ`: the world-space Z component of the center of the docking region.
    - e.g. `-z -36`
    - This argument is optional. The default is 0, and the expected range of values is -infinity to +infinity.
- `-w` or `--dockingRegionWidth`: the world-space width of the docking region.
    - e.g. `-w 41.25`
    - This argument is optional. The default is 2.4, and the expected range of values is > 0.
- `-e` or `--emitterUVName`: the name of the UV data channel (known as a primvar in USD) where the computed emitter UVs will be stored.
    - e.g. `-e primvars:emissionUV`
    - This argument is optional, the default is `primvars:emissionUV`
- `--emitterUVExponent`: the exponent used for the emitter UV falloff. Can be used to artistically control how far out the diffuse reflection colors appear.
    - e.g. `--emitterUVExponent 6.0`
    - This argument is optional. The default is `6.0`, and the expected range of values is >= `2.0`
- `-s` or `--sampleCount`: the number of random samples used during emitter UV calculation. Decreasing this number can speed up computation time, at the cost of potential artifacts.
    - e.g. `-s 5000`
    - This argument is optional. The default value is `5000` and the expected range of values is > 0
- `-a` or `--attenuationUVName`: the name of the UV data channel (known as a primvar in USD) where the computed attenuation UVs will be stored.

    - e.g. `-a primvars:attenuationUV`
    - This argument is optional, the default is `primvars:attenuationUV`
- `--attenuationUStart`, `--attenuationUEnd`, `--attenuationVStart`, `--attenuationVEnd`: the attenuation texture measurements which control how the attenuation texture is top-down projected onto the geometry. Please see the “Measuring Attenuation Textures” section above.
    - e.g. `--attenuationUStart 0.25 --attenuationUEnd 0.75 --attenuationVStart 0 --attenuationVEnd 1.0`
    - These arguments are optional. If they are not provided, values corresponding to Reality Composer Pro’s default provided attenuation texture will be used. The expected range of these values are 0 to 1.
- `-v` or `--verbose`: prints out script completion percentage, and helpful information for setting up the resulting USD file in Reality Composer Pro.
    - e.g. `-v true`
    - This argument is optional. The default is `true`.

4. Using the Python Script Output

Once you’ve used the Python script, you will need to replace the original USD file with the generated output USD file in Reality Composer Pro (RCP).

The custom material used for the object receiving diffuse reflection can make use of a “Reflection Diffuse (RealityKit)” node, which takes both an “Emitter UV” and an “Attenuation UV” input. You will need to connect a “Primvar Reader” node to each one, and they will need to use the name provided to the `--emitterUVName` and `--attenuationUVName` arguments listed above. So for example, if you specified “primvars:emitterUV”, then the Primvar Reader Node’s “Varname” input should be “emitterUV”.