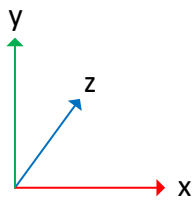
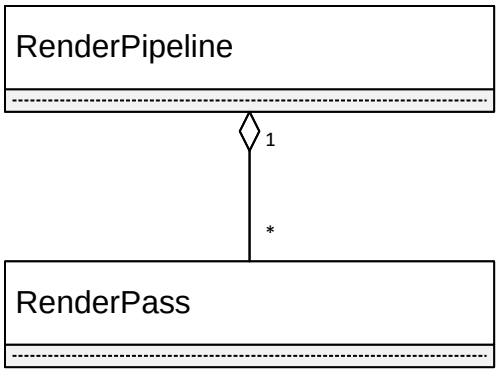
# The Render Engine Architecture in a Nutshell

**Coordinate System: Left-Handed**




**RenderPipeline** is the class that holds everything together and updates multiple RenderPasses per frame. It also provides *put*- and *fetch*-instructions to pass arbitrary data between render passes.

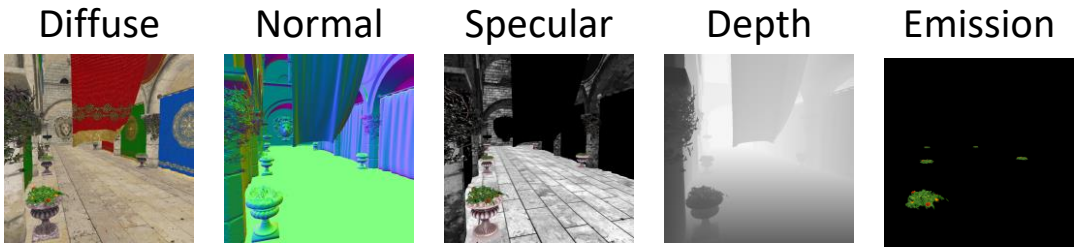



## Render Pass

A specific render pass can derive from this class to implement rendering features. The following passes are currently used for GI (global illumination) in this order:

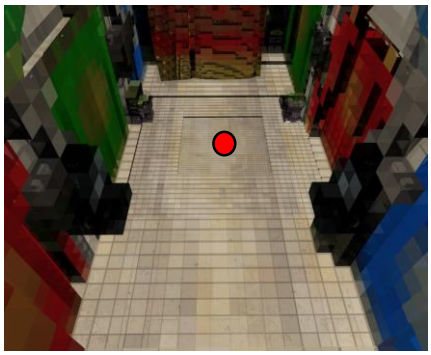
## SceneGeometryPass

Renders the geometry properties (from camera perspective) of the scene to **G-Buffers** (Geometry Buffers), e.g. the following:



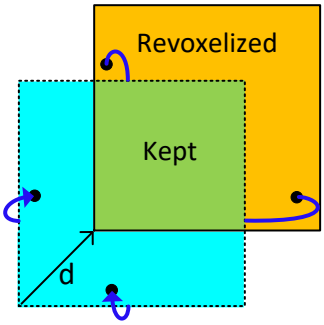

## VoxelizationPass

Voxelizes required geometry properties (currently only opacity) and stores them in a 3D voxel clipmap which is built around the camera position. The following figure shows multiple voxelized clipmap levels around the red point (voxelized color in this case).



The voxelization process only takes place when
**a)** the camera moves or
**b)** an object transform changes.

For **a)** only the required part will be revoxelized (using toroidal addressing) when the camera moves by a vector $d$ as the following figure shows:




For **b)** the AABB of the object of the last frame and current frame are added to a list of regions that need to be revoxelized. Overlapping AABBs are merged.

## ShadowMapPass

Generates shadow maps for directional lights and makes them available to subsequent passes.

## RadianceInjectionPass

Injects radiance into voxels. Currently this is done by voxelizing all voxel clipmap levels using directional lights, geometry properties and shadow maps. The number of updated clipmap levels per frame may be *one* or *all* and is selected in the editor.

## WrapBorderPass

Since all voxel faces and clipmap levels are stored in one big 3D texture for each voxel attribute (interleaved) GL_REPEAT can't be used as the texture wrapping mode. Thus to avoid bleeding and to ensure correct interpolation (at borders) the border needs to be extended by one and the other side copied over to get GL_REPEAT functionality. That's what this render pass does.

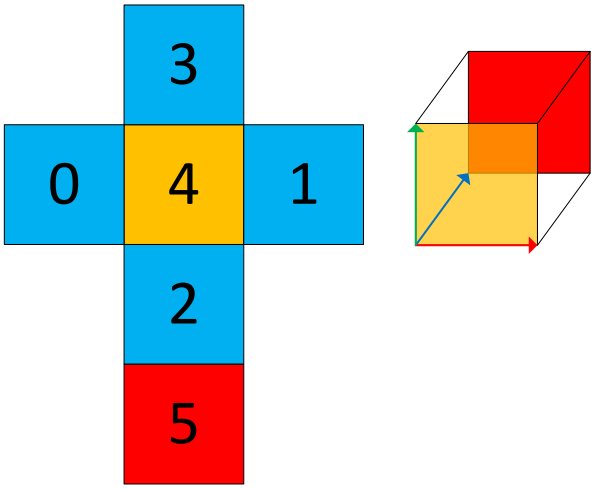
## GIPass

Computes illumination (direct + indirect) for each G-Buffer pixel. *Direct lighting* is computed by standard surface shading techniques (selection between Blinn-Phong and Cook-Torrance BRDFs, Shadow Maps with PCF for soft shadows with constant penumbra size). *Indirect Lighting* is computed with voxel cone tracing:

**Diffuse**: Multiple cones are traced to gather diffuse indirect radiance (approximation of the hemisphere with cones using spherical 3D design points). Currently 16 sphere points are used ($\sim 8$ hemisphere points) for default quality and 32 may be enabled in the shader defines for high quality.

**Specular**: The view vector is reflected at the surface normal and traced to compute the specular indirect contribution.

**Anisotropic Voxel Indexing**

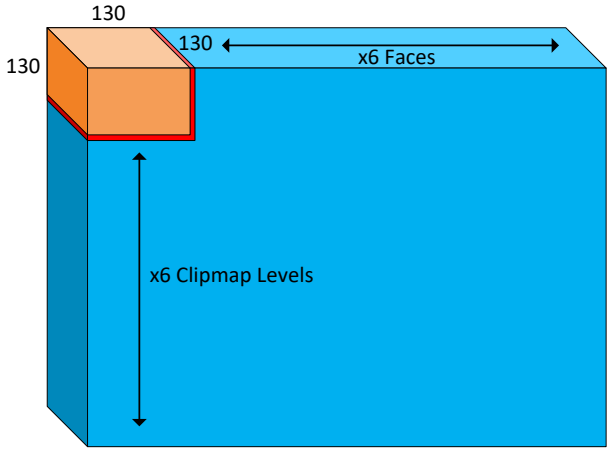



**Voxel Clipmap**

The clipmap has no pyramid in the implementation because of the complexity implications resulting from the data layout and a performance impact with little quality gain.

Data Layout

The default resolution is $128^3$ however the border needs to be extended by one to ensure correct interpolation at the border of the texture and avoid bleeding to neighboring textures so a resolution of $130^3$ is required. All six faces are stored interleaved in the X-direction, all clipmap levels interleaved in the Y-direction of one big 3D texture ($780 \times 780 \times 130$):



**Note**: For the sake of simplicity the border is also extended in the Z-direction however there is actually no need for it since GL_REPEAT can be used in this case.

### Voxelization Projection Planes

For each main scene axis a view projection matrix is computed. This is necessary because during voxelization the *dominant axis* is selected (axis which yields the highest projected area of the triangle as shown in the following figure). The resolution of the viewport should usually coincide with the resolution of the voxel texture ($128^2$ to voxelize a $128^3$ region). In this case however only a subset is voxelized when the camera or objects move and can lead to missed fragments due to floating point imprecision because the resolution is too tight to include them. A simple yet effective solution for this problem is to extend the resolution by two in each dimension to get a resolution of $130^2$. This will include all fragments that would otherwise be missed.