docs.godotengine.org

Spatial shaders

12–15 minutes

Spatial shaders are used for shading 3D objects. They are the most complex type of shader Godot offers. Spatial shaders are highly configurable with different render modes and different rendering options (e.g. Subsurface Scattering, Transmission, Ambient Occlusion, Rim lighting etc). Users can optionally write vertex, fragment, and light processor functions to affect how objects are drawn.

Render modes¶

Render mode	Description
blend_mix	Mix blend mode (alpha is transparency), default.
blend_add	Additive blend mode.
blend_sub	Subtractive blend mode.
blend_mul	Multiplicative blend mode.
depth_draw_opaque	Only draw depth for opaque geometry (not transparent).
depth_draw_always	Always draw depth (opaque and transparent).
depth_draw_never	Never draw depth.
depth_draw_alpha_prepass	Do opaque depth pre-pass for transparent geometry.
depth_test_disable	Disable depth testing.
cull_front	Cull front-faces.
cull_back	Cull back-faces (default).

Render mode	Description
cull_disabled	Culling disabled (double sided).
unshaded	Result is just albedo. No lighting/shading happens in material.
diffuse_lambert	Lambert shading for diffuse (default).
diffuse_lambert_wrap	Lambert wrapping (roughness dependent) for diffuse.
diffuse_oren_nayar	Oren Nayar for diffuse.
diffuse_burley	Burley (Disney PBS) for diffuse.
diffuse_toon	Toon shading for diffuse.
specular_schlick_ggx	Schlick-GGX for specular (default).
specular_blinn	Blinn for specular (compatibility).
specular_phong	Phong for specular (compatibility).
specular_toon	Toon for specular.
specular_disabled	Disable specular.
skip_vertex_transform	VERTEX/NORMAL/etc. need to be transformed manually in vertex function.
world_vertex_coords	VERTEX/NORMAL/etc. are modified in world coordinates instead of local.
ensure_correct_normals	Use when non-uniform scale is applied to mesh.
vertex_lighting	Use vertex-based lighting.
shadows_disabled	Disable computing shadows in shader.
ambient_light_disabled	Disable contribution from ambient light and radiance map.

Render mode	Description
shadow_to_opacity	Lighting modifies the alpha so shadowed areas are opaque and non-shadowed areas are transparent. Useful for overlaying shadows onto a camera feed in AR.

Built-ins¶

Values marked as "in" are read-only. Values marked as "out" are for optional writing and will not necessarily contain sensible values. Values marked as "inout" provide a sensible default value, and can optionally be written to. Samplers are not subjects of writing and they are not marked.

Global built-ins¶

Global built-ins are available everywhere, including custom functions.

Built-in	Description
in float **TIME**	Global time, in seconds.

Vertex built-ins¶

Vertex data (`VERTEX`, `NORMAL`, `TANGENT`, `BITANGENT`) are presented in local model space. If not written to, these values will not be modified and be passed through as they came.

They can optionally be presented in world space by using the *world_vertex_coords* render mode.

Users can disable the built-in modelview transform (projection will still happen later) and do it manually with the following code:

```
shader_type spatial;
render_mode skip_vertex_transform;

void vertex() {
    VERTEX = (MODELVIEW_MATRIX * vec4(VERTEX, 1.0)).xyz;
    NORMAL = normalize((MODELVIEW_MATRIX * vec4(NORMAL, 0.0)).xyz);
    // same as above for binormal and tangent, if normal mapping is
used
}
```

Other built-ins, such as UV, UV2 and COLOR, are also passed through to the fragment function if not modified.

Users can override the modelview and projection transforms using the `POSITION` built-in. When `POSITION` is used, the value from `VERTEX` is ignored and projection does not happen. However, the value passed to the fragment shader still comes from `VERTEX`.

For instancing, the INSTANCE_CUSTOM variable contains the instance custom data. When using particles, this information is usually:

- **x**: Rotation angle in radians.
- **y**: Phase during lifetime (0 to 1).
- **z**: Animation frame.

This allows you to easily adjust the shader to a particle system using default particles material. When writing a custom particle shader, this value can be used as desired.

Built-in	Description
in vec2 **VIEWPORT_SIZE**	Size of viewport (in pixels).
inout mat4 **WORLD_MATRIX**	Model space to world space transform.
in mat4 **INV_CAMERA_MATRIX**	World space to view space transform.
inout mat4 **PROJECTION_MATRIX**	View space to clip space transform.
in mat4 **CAMERA_MATRIX**	View space to world space transform.
inout mat4 **MODELVIEW_MATRIX**	Model space to view space transform (use if possible).
inout mat4 **INV_PROJECTION_MATRIX**	Clip space to view space transform.
in vec3 **NODE_POSITION_WORLD**	Node position, in world space.
in vec3 **NODE_POSITION_VIEW**	Node position, in view space.
in vec3 **CAMERA_POSITION_WORLD**	Camera position, in world space.
in vec3 **CAMERA_DIRECTION_WORLD**	Camera direction, in world space.

Built-in	Description
inout vec3 **VERTEX**	Vertex in local coordinates.
in int **VERTEX_ID**	The index of the current vertex in the vertex buffer. Not supported in GLES2 (returns 0).
out vec4 **POSITION**	If written to, overrides final vertex position.
inout vec3 **NORMAL**	Normal in local coordinates.
inout vec3 **TANGENT**	Tangent in local coordinates.
inout vec3 **BINORMAL**	Binormal in local coordinates.
out float **ROUGHNESS**	Roughness for vertex lighting.
inout vec2 **UV**	UV main channel.
inout vec2 **UV2**	UV secondary channel.
in bool **OUTPUT_IS_SRGB**	`true` when calculations happen in sRGB color space (`true` in GLES2, `false` in GLES3).
inout vec4 **COLOR**	Color from vertices.
inout float **POINT_SIZE**	Point size for point rendering.
in int **INSTANCE_ID**	Instance ID for instancing. Not supported in GLES2 (returns 0).
in vec4 **INSTANCE_CUSTOM**	Instance custom data (for particles, mostly).

Note

`MODELVIEW_MATRIX` combines both the `WORLD_MATRIX` and `INV_CAMERA_MATRIX` and is better suited when floating point issues may arise. For example, if the object is very far away from the world origin, you may run into floating point issues when using the seperated `WORLD_MATRIX` and `INV_CAMERA_MATRIX`.

Fragment built-ins¶

The default use of a Godot fragment processor function is to set up the material properties of your object and to let the built-in renderer handle the final shading. However, you are not required to use all these properties, and if you don't write to them, Godot will optimize away the corresponding functionality.

Built-in	Description
in vec2 **VIEWPORT_SIZE**	Size of viewport (in pixels).
in vec4 **FRAGCOORD**	Coordinate of pixel center in screen space. `xy` specifies position in window, `z` specifies fragment depth if `DEPTH` is not used. Origin is lower-left.
in mat4 **WORLD_MATRIX**	Model space to world space transform.
in mat4 **INV_CAMERA_MATRIX**	World space to view space transform.
in mat4 **CAMERA_MATRIX**	View space to world space transform.
in mat4 **PROJECTION_MATRIX**	View space to clip space transform.
in mat4 **INV_PROJECTION_MATRIX**	Clip space to view space transform.
in vec3 **NODE_POSITION_WORLD**	Node world space position.
in vec3 **NODE_POSITION_VIEW**	Node view space position.
in vec3 **CAMERA_POSITION_WORLD**	Camera world space position.
in vec3 **CAMERA_DIRECTION_WORLD**	Camera world space direction.
in vec3 **VERTEX**	Vertex that comes from vertex function (default, in view space).
in vec3 **VIEW**	Vector from camera to fragment position (in view space).
in bool **FRONT_FACING**	`true` if current face is front face.

Built-in	Description
inout vec3 **NORMAL**	Normal that comes from vertex function (default, in view space).
inout vec3 **TANGENT**	Tangent that comes from vertex function.
inout vec3 **BINORMAL**	Binormal that comes from vertex function.
out vec3 **NORMALMAP**	Set normal here if reading normal from a texture instead of NORMAL.
out float **NORMALMAP_DEPTH**	Depth from variable above. Defaults to 1.0.
in vec2 **UV**	UV that comes from vertex function.
in vec2 **UV2**	UV2 that comes from vertex function.
in bool **OUTPUT_IS_SRGB**	`true` when calculations happen in sRGB color space (`true` in GLES2, `false` in GLES3).
in vec4 **COLOR**	COLOR that comes from vertex function.
out vec3 **ALBEDO**	Albedo (default white).
out float **ALPHA**	Alpha (0..1); if written to, the material will go to the transparent pipeline.
out float **ALPHA_SCISSOR**	If written to, values below a certain amount of alpha are discarded.
out float **METALLIC**	Metallic (0..1).
out float **SPECULAR**	Specular. Defaults to 0.5, best not to modify unless you want to change IOR.
out float **ROUGHNESS**	Roughness (0..1).
out float **RIM**	Rim (0..1). If used, Godot calculates rim lighting.

Built-in	Description
out float **RIM_TINT**	Rim Tint, goes from 0 (white) to 1 (albedo). If used, Godot calculates rim lighting.
out float **CLEARCOAT**	Small added specular blob. If used, Godot calculates Clearcoat.
out float **CLEARCOAT_GLOSS**	Gloss of Clearcoat. If used, Godot calculates Clearcoat.
out float **ANISOTROPY**	For distorting the specular blob according to tangent space.
out vec2 **ANISOTROPY_FLOW**	Distortion direction, use with flowmaps.
out float **SSS_STRENGTH**	Strength of Subsurface Scattering. If used, Subsurface Scattering will be applied to object.
out vec3 **TRANSMISSION**	Transmission mask (default 0,0,0). Allows light to pass through object. Only applied if used.
out vec3 **EMISSION**	Emission color (can go over 1,1,1 for HDR).
out float **AO**	Strength of Ambient Occlusion. For use with pre-baked AO.
out float **AO_LIGHT_AFFECT**	How much AO affects lights (0..1; default 0).
sampler2D **SCREEN_TEXTURE**	Built-in Texture for reading from the screen. Mipmaps contain increasingly blurred copies.
sampler2D **DEPTH_TEXTURE**	Built-in Texture for reading depth from the screen. Must convert to linear using INV_PROJECTION.
out float **DEPTH**	Custom depth value (0..1). If DEPTH is being written to in any shader branch, then you are responsible for setting the DEPTH for **all** other branches. Otherwise, the graphics API will leave them uninitialized.

Built-in	Description
in vec2 **SCREEN_UV**	Screen UV coordinate for current pixel.
in vec2 **POINT_COORD**	Point Coordinate for drawing points with POINT_SIZE.

Light built-ins¶

Writing light processor functions is completely optional. You can skip the light function by setting render_mode to `unshaded`. If no light function is written, Godot will use the material properties written to in the fragment function to calculate the lighting for you (subject to the render_mode).

To write a light function, assign something to `DIFFUSE_LIGHT` or `SPECULAR_LIGHT`. Assigning nothing means no light is processed.

The light function is called for every light in every pixel. It is called within a loop for each light type.

Below is an example of a custom light function using a Lambertian lighting model:

```
void light() {
    DIFFUSE_LIGHT += clamp(dot(NORMAL, LIGHT), 0.0, 1.0) * ATTENUATION
* ALBEDO;
}
```

If you want the lights to add together, add the light contribution to `DIFFUSE_LIGHT` using `+=`, rather than overwriting it.

Warning

In GLES2, lights will always be added together even if you override `DIFFUSE_LIGHT` using `=`. This is due to lighting being computed in multiple passes (one for each light), unlike GLES3.

Warning

The `light()` function won't be run if the `vertex_lighting` render mode is enabled, or if **Rendering > Quality > Shading > Force Vertex Shading** is enabled in the Project Settings. (It's enabled by default on mobile platforms.)

Built-in	Description
in float **TIME**	Elapsed total time in seconds.

Built-in	Description
in vec2 **VIEWPORT_SIZE**	Size of viewport (in pixels).
in vec4 **FRAGCOORD**	Coordinate of pixel center in screen space. `xy` specifies position in window, `z` specifies fragment depth if `DEPTH` is not used. Origin is lower-left.
in mat4 **WORLD_MATRIX**	Model space to world space transform.
in mat4 **INV_CAMERA_MATRIX**	World space to view space transform.
in mat4 **CAMERA_MATRIX**	View space to world space transform.
in mat4 **PROJECTION_MATRIX**	View space to clip space transform.
in mat4 **INV_PROJECTION_MATRIX**	Clip space to view space transform.
in vec3 **NORMAL**	Normal vector, in view space.
in vec2 **UV**	UV that comes from vertex function.
in vec2 **UV2**	UV2 that comes from vertex function.
in vec3 **VIEW**	View vector, in view space.
in vec3 **LIGHT**	Light Vector, in view space.
in vec3 **ATTENUATION**	Attenuation based on distance or shadow.
in bool **OUTPUT_IS_SRGB**	`true` when calculations happen in sRGB color space (`true` in GLES2, `false` in GLES3).
in vec3 **ALBEDO**	Base albedo.
in vec3 **LIGHT_COLOR**	Color of light multiplied by `energy * PI`. The `PI` multiplication is present because physically-based lighting models include a division by `PI`.

Built-in	Description
out float **ALPHA**	Alpha (0..1); if written to, the material will go to the transparent pipeline.
in float **ROUGHNESS**	Roughness.
in vec3 **TRANSMISSION**	Transmission mask from fragment function.
out vec3 **DIFFUSE_LIGHT**	Diffuse light result.
out vec3 **SPECULAR_LIGHT**	Specular light result.