docs.godotengine.org

CanvasItem shaders

8–10 minutes

CanvasItem shaders are used to draw all 2D elements in Godot. These include all nodes that inherit from CanvasItems, and all GUI elements.

CanvasItem shaders contain less built-in variables and functionality than Spatial shaders, but they maintain the same basic structure with vertex, fragment, and light processor functions.

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.
blend_premul_alpha	Pre-multiplied alpha blend mode.
blend_disabled	Disable blending, values (including alpha) are written as-is.
unshaded	Result is just albedo. No lighting/shading happens in material.
light_only	Only draw on light pass.
skip_vertex_transform	VERTEX/NORMAL/etc need to be transformed manually in vertex function.

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 since the engine has started, in seconds (always positive). It's subject to the rollover setting (which is 3,600 seconds by default). It's not affected by time_scale or pausing, but you can override the TIME variable's time scale by calling `VisualServer.set_shader_time_scale()` with the desired time scale factor as parameter (`1.0` being the default).

Vertex built-ins¶

Vertex data (VERTEX) is presented in local space (pixel coordinates, relative to the camera). If not written to, these values will not be modified and be passed through as they came.

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

```
shader_type canvas_item;
render_mode skip_vertex_transform;

void vertex() {

    VERTEX = (WORLD_MATRIX * (EXTRA_MATRIX * vec4(VERTEX, 0.0,
1.0))).xy;
}
```

Note

`WORLD_MATRIX` is actually a modelview matrix. It takes input in local space and transforms it into view space.

In order to get the world space coordinates of a vertex, you have to pass in a custom uniform like so:

```
material.set_shader_param("global_transform", get_global_transform())
```

Then, in your vertex shader:

```
uniform mat4 global_transform;
varying vec2 world_position;

void vertex(){
    world_position = (global_transform * vec4(VERTEX, 0.0, 1.0)).xy;
}
```

`world_position` can then be used in either the vertex or fragment functions.

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

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.

Built-in	Description
in mat4 **WORLD_MATRIX**	Image space to view space transform.
in mat4 **EXTRA_MATRIX**	Extra transform.
in mat4 **PROJECTION_MATRIX**	View space to clip space transform.
in int **INSTANCE_ID**	Instance ID for instancing. Not supported in GLES2 (returns 0).
in vec4 **INSTANCE_CUSTOM**	Instance custom data.
in bool **AT_LIGHT_PASS**	`true` if this is a light pass.
inout vec2 **VERTEX**	Vertex, in image space.
in int **VERTEX_ID**	The index of the current vertex in the vertex buffer. Not supported in GLES2 (returns 0).

Built-in	Description
in vec2 **TEXTURE_PIXEL_SIZE**	Normalized pixel size of default 2D texture. For a Sprite with a texture of size 64x32px, **TEXTURE_PIXEL_SIZE** = `vec2(1/64, 1/32)`
inout vec2 **UV**	Texture coordinates.
inout vec4 **COLOR**	Color from vertex primitive.
in vec4 **MODULATE**	Final modulate color. If used, **COLOR** will not be multiplied by modulate automatically after the fragment function.
inout float **POINT_SIZE**	Point size for point drawing.

Fragment built-ins¶

Certain Nodes (for example, Sprites) display a texture by default. However, when a custom fragment function is attached to these nodes, the texture lookup needs to be done manually. Godot does not provide the texture color in the `COLOR` built-in variable; to read the texture color for such nodes, use:

```
COLOR = texture(TEXTURE, UV);
```

This differs from the behavior of the built-in normal map. If a normal map is attached, Godot uses it by default and assigns its value to the built-in `NORMAL` variable. If you are using a normal map meant for use in 3D, it will appear inverted. In order to use it in your shader, you must assign it to the `NORMALMAP` property. Godot will handle converting it for use in 2D and overwriting `NORMAL`.

```
NORMALMAP = texture(NORMAL_TEXTURE, UV).rgb;
```

Built-in	Description
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.
inout vec3 **NORMAL**	Normal read from **NORMAL_TEXTURE**. Writable.

Built-in	Description
out vec3 **NORMALMAP**	Configures normal maps meant for 3D for use in 2D. If used, overwrites **NORMAL**.
inout float **NORMALMAP_DEPTH**	Normalmap depth for scaling.
in vec2 **UV**	UV from vertex function.
inout vec4 **COLOR**	Color from vertex function and output fragment color. If unused, will be set to **TEXTURE** color.
in vec4 **MODULATE**	Final modulate color. If used, **COLOR** will not be multiplied by modulate automatically after the fragment function.
in sampler2D **TEXTURE**	Default 2D texture.
in sampler2D **NORMAL_TEXTURE**	Default 2D normal texture.
in vec2 **TEXTURE_PIXEL_SIZE**	Normalized pixel size of default 2D texture. For a Sprite with a texture of size 64x32px, **TEXTURE_PIXEL_SIZE** = `vec2(1/64, 1/32)`
in vec2 **SCREEN_UV**	Screen UV for use with **SCREEN_TEXTURE**.
in vec2 **SCREEN_PIXEL_SIZE**	Size of individual pixels. Equal to inverse of resolution.
in vec2 **POINT_COORD**	Coordinate for drawing points.
in bool **AT_LIGHT_PASS**	`t`rue if this is a light pass.
in sampler2D **SCREEN_TEXTURE**	Screen texture, mipmaps contain gaussian blurred versions.

Light built-ins¶

Light processor functions work differently in 2D than they do in 3D. In CanvasItem shaders, the shader is called once for the object being drawn, and then once for each

light touching that object in the scene. Use render_mode `unshaded` if you do not want any light passes to occur for that object. Use render_mode `light_only` if you only want light passes to occur for that object; this can be useful when you only want the object visible where it is covered by light.

When the shader is on a light pass, the `AT_LIGHT_PASS` variable will be `true`.

Built-in	Description
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 vec3 **NORMAL**	Input Normal. Although this value is passed in, **normal calculation still happens outside of this function**.
in vec2 **UV**	UV from vertex function, equivalent to the UV in the fragment function.
in vec4 **COLOR**	Input Color. This is the output of the fragment function (with final modulation applied, if **MODULATE** is not used in any function of the shader).
in vec4 **MODULATE**	Final modulate color. If used, **COLOR** will not be multiplied by modulate automatically after the fragment function.
sampler2D **TEXTURE**	Current texture in use for CanvasItem.
in vec2 **TEXTURE_PIXEL_SIZE**	Normalized pixel size of default 2D texture. For a Sprite with a texture of size 64x32px, **TEXTURE_PIXEL_SIZE** = `vec2(1/64, 1/32)`
in vec2 **SCREEN_UV**	**SCREEN_TEXTURE** Coordinate (for using with screen texture).
in vec2 **POINT_COORD**	UV for Point Sprite.
inout vec2 **LIGHT_VEC**	Vector from light to fragment in local coordinates. It can be modified to alter illumination direction when normal maps are used.

Built-in	Description
inout vec2 **SHADOW_VEC**	Vector from light to fragment in local coordinates. It can be modified to alter shadow computation.
inout float **LIGHT_HEIGHT**	Height of Light. Only effective when normals are used.
inout vec4 **LIGHT_COLOR**	Color of Light.
in vec2 **LIGHT_UV**	UV for Light texture.
out vec4 **SHADOW_COLOR**	Shadow Color of Light.
inout vec4 **LIGHT**	Value from the Light texture and output color. Can be modified. If not used, the light function is ignored.