# SittingDuck's Advanced CRT Shader for GameMaker

A lightweight and flexible CRT shader that aims to be scalable and easy to implement.

### To implement with your existing game:

- Place a preset object in the first room of your game or create & destroy one via code in your game's menu.
- Ensure that your game's aspect ratio matches that of the preset object.
- That's it!

### Advanced use:

- All parameters of the shader are addressable from any object just by referencing the persistent CRT object.
  - For example: "crt.glow_strength += 0.1"
  - Just make sure to update the shader uniforms afterward with "crt.update_uniforms()"
- Learn to use effects that retro game developers used with CRT TVs in mind.
  - Exploit interlacing, blending, or phosphors for your game's benefit!

### Notes:

- This asset expects your game to fill the entire application surface. It is then scaled down to the appropriate size and aspect ratio.
  - As an example, a game targeting 4:3 might have a view of size 320x240, with a viewport that must be set to stetch to fill the whole screen at something like 1920x1080.
  - 3D games can simply read the CRT setting, like so: `matrix_build_projection_perspective_fov( fov, crt.aspect_ratio, znear, zfar)`
- GUI elements in your game can be drawn using the "Draw GUI" event like normal.
  - However, be aware that pixelated GUI graphics must be natively-sized (without prebaked scaling).
  - You can also choose to draw a GUI over top the CRT screen, in which case you should expect the GUI size to match the window size.

## Disclaimer:

- This asset is not intended to produce a simple stylized effect. It aims to accurately and faithfully recreate what pixel art used to look like on old tube TVs.
  - The optimal settings will depend on your display technology, including resolution, aspect ratio, and subpixel layout. It is *strongly* recommended that you expose the CRT settings to your players in your game's menu, so that they can tweak it to their liking.
- If you want something a bit simpler that doesn't require any parameter tuning (e.g., for game jams), I personally recommend BJÖRTFX by ZIK: https://zikbakguru.itch.io/bjrtfx

## Methods:

- `crt.resize_surfaces()`
  - Ensures that the view and application surfaces are the size that the CRT shader expects.
  - This method is more useful when integrating the shader with an existing game, at the start of every room.
- `crt.update_uniforms()`
  - This function simply passes all of the preset object's parameters into the shader as uniforms.
  - It should be called whenever these parameters change.
- `crt.crt_apply()`
  - This method draws the application surface with the CRT shader applied.
  - It should typically be done in the Draw GUI End event.
- `crt.set_shader()`
  - This method allows you to change the shader being used. It also updates the uniforms for you.
  - Included are 3 options:
    + `shd_crt` - This is the full CRT shader with high quality blur and advanced interpolation.
    + `shd_crt_fast` - This is a lower-quality but still fully-featured shader that is easier to run on low-spec machines, using only half as many texture lookups per pixel.
    + `shd_raw` - This is a shader with curvature and nothing else. It's useful for comparing unfiltered to filtered gameplay.
- Global: `get_mouse_coords()`
  - This method returns the in-game mouse coordinates, correcting for the screen curvature and aspect ratio.
  - It was added to restore mouse_x and mouse_y functionality for ease of integrating with existing projects.

## Shader parameters:

- `output_width` - [Any integer value]
  - The width of the game's output in pixels (typically the device resolution)
  - If –1, the width of the game window is used to determine the value
- `output_height` - [Any integer value]
  - The height of the game's output in pixels (typically the device resolution)
  - If –1, the height of the game window is used to determine the value.
- `game_width` - [Any positive integer value]
  - The width of the retro game in pixels (e.g., 320 for PSX games)
- `game_height` - [Any positive integer value]
  - The height of the retro game in pixels (e.g., 240 for PSX games)
- `do_integer_scale` - [Boolean]
  - Whether or not to scale in whole pixels only (typically with a black border)
  - Useful for lower-resolution devices where non-uniform scaling can stand out.
- `screen_aspect_ratio` - [Expression, evaluates to real]
  - How much the game should be squished or stretched horizontally.
  - Can be input as a fraction (4/3) or as a real value (1.33)
  - Retro consoles and arcade games often had non-square pixels, meaning the pixels were wider than they were tall or vice versa. The shader will take care of this automatically; all you really need to decide is the final aspect ratio of the output. For a retro feel, go with 4/3, or for modern games, try 16/9.
- `screen_do_tate` - [Boolean]
  - Whether or not to rotate the phosphor mask and scanlines (for portrait games)
  - Should only be used with the RGBx mask, or with the mask disabled.
- `screen_curvature_amount` - [Any real value between 0.0 and 1.0]
  - How much curvature the CRT screen will have. Less is more in most cases.
- `glow_spread` - [Any real value between 0.0 and 5.0]
  - How wide the glow effect will spread in pixels
- `glow_gamma` - [Any real value between 1.0 and 5.0]
  - The shape of the ramp function the glow fadeout is based on
  - Larger values are "rounder" while lower values are "sharper"
- `glow_strength` - [Any real value between 0.0 and 5.0]
  - How strong the glow effect is
- `glow_over_mask` - [Any real value between 0.0 and 1.0]
  - How much the glow effect should bloom over the black parts of the phosphor mask
- `image_softness` - [Any real value between 0.0 and 1.0]
  - How blurry to make the image interpolation
  - 0.0 = quintic interpolation, 1.0 = bilinear interpolation
- `image_noise_amount` - [Any real value between 0.0 and 1.0]

  - How much TV static to add to the image
- `image_deconvergence` - [Any real value between −1.0 and 1.0]
  - How much to separate the RGB subsignals of the image
  - This produces an effect similar to chromatic aberration in filmography.
- `input_gamma` - [Any real value between 1.0 and 5.0]
  - The gamma of the input signal (should almost always be 2.2)
- `output_gamma` - [Any real value between 1.0 and 5.0]
  - The gamma of the output signal (should almost always be 2.2)
- `mask_type` - [Sprite asset]
  - Which phosphor mask pattern to use (see Figure 1)
    + `RGBx` - The most basic pattern, but also the darkest (requires brightness adjustment). This is the only mask you should use with a `mask_scale` other than 1.0
    + GM - This mask takes advantage of the host display's subpixel layout to increase brightness. Best suited for low-res devices with `mask_scale` of 1.0
    + RYCB - This mask takes advantage of the host display's subpixel layout to increase brightness. Best suited for high-res devices with `mask_scale` of 1.0
    + YB - Inverse of GM for devices with a nonstandard subpixel layout. Best suited for low-res devices with `mask_scale` of 1.0
    + RMCG - Inverse of RYCB for devices with a nonstandard subpixel layout. Best suited for high-res devices with `mask_scale` of 1.0
- `mask_strength` - [Any real value between 0.0 and 1.0]
  - How strongly the mask will affect the image
  - 1.0 is technically speaking the most accurate, but also darkens the image the most.
- `mask_scale` - [Any real value]
  - The size of the phosphor mask
  - Should be set to 1.0 whenever possible on LCD panels, but for OLEDs anything goes.
- `mask_rgbx_repeat_on_threes` - [Boolean]
  - Whether or not to crop the black column when using the `RGBx` mask
  - This can increase brightness, but also raises the TVL of the CRT screen
- `mask_do_mirror` - [Boolean]
  - Whether or not to flip the phosphor mask (for nonstandard displays)
- `mask_convert_to_shadow` - [Boolean]
  - Whether or not to convert the default aperture grille into a shadow mask
  - This option is recommended for low-res devices, or if aiming to simulate the look of a VGA monitor.
- `mask_slot_strength` - [Any real value between 0.0 and 1.0]
  - How strongly to apply a slot mask effect (if at all)
  - Recommended for high-res devices, or when trying to simulate the look of a consumer TV set or arcade monitor.
- `mask_slot_height` - [Integer value: 0, 1, or 2]

  - The height of the individual slots for the slot mask effect
- `scanline_strength` - [Any real value between 0.0 and 5.0]
  - How dark the area between electron beams is
- `scanline_bright_fade` - [Any real value between 0.0 and 1.0]
  - How much to allow scanlines to "grow" or "bleed" on bright parts of the image
  - This is often referred to as "beam dynamics," and was a real phenomenon on CRTs where darker scanlines were thinner than bright ones.
- `scanline_shape` - [Any real value between 1.0 and 2.0]
  - The shape of the electron beam
  - 1.0 = linear, like a triangle wave, 2.0 = quadratic, a round shape
- `border_width` - [Any real value between 0.0 and 10.0]
  - How wide to make the border outside of the CRT screen
  - Setting this to 0.0 effectively removes the border.
- `border_brightness` - [Any real value between 0.0 and 1.0]
  - How bright the screen reflection on the border should be
- `viewport_zoom` - [Any real value]
  - How much to zoom in on the CRT screen (useful for screenshots)
- `viewport_shift_x` - [Any real value]
  - How much to shift the CRT screen horizontally (perhaps to make room for a GUI?)
- `viewport_shift_y` - [Any real value]
  - How much to shift the CRT screen vertically (perhaps to make room for a GUI?)
- `post_brightness` - [Any real value]
  - How much to alter the brightness right before displaying to the screen
  - This should usually be your last resort; try changing other parameters first.
- `shader` - [Shader asset]
  - Which CRT shader to use for drawing. You have 3 options:
    + `shd_crt` - Fully-featured, high quality shader. Expensive.
    + `shd_crt_fast` - Still feature-complete, but lower visual quality. Faster.
    + `shd_raw` - Only handles curvature and aspect ratio. For comparisons.
- `gui_user_event` - [Array of GM.EventName, GM.EventNumber]
  - Event used for game objects to draw to the GUI.
  - Any drawing done in this event is captured, so it will appear on the CRT screen.
  - The CRT object's draw events will never be captured.
  - If you want to draw your GUI on top of the CRT screen, change this to a user event.
  - For improved performance, use a user event and move all your GUI drawing to it!
- `crt_draw_event` - [Choice of "Draw GUI Begin", "Draw GUI", or "Draw GUI End"]
  - When to capture all drawing, apply the shader, and send to the display
  - Use "Draw GUI Begin" if you want to draw a GUI on top of the CRT screen.
  - Use "Draw GUI End" in most other cases, with the goal of capturing GUI elements.

- `do_overlay` - [Boolean]
  - Whether or not to draw an overlay over or under the CRT screen
- `overlay_over_screen` - [Boolean]
  - Whether or not the overlay should be in front of the screen or behind it
- `overlay_image` - [Sprite asset]
  - The image you will be using as an overlay
  - If drawing in front of the CRT screen, there needs to be transparency in the image.

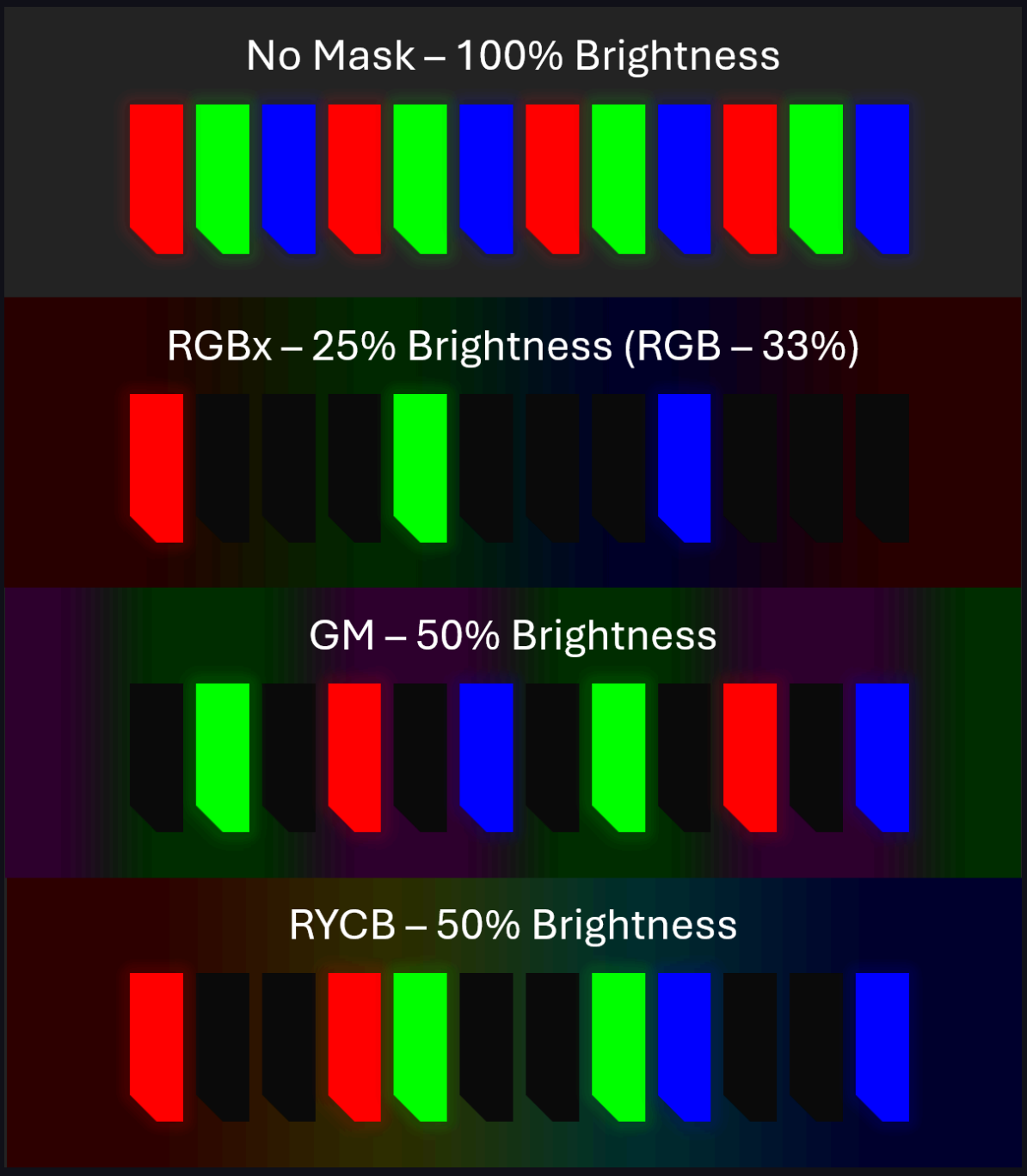


Figure 1: A demonstration of the main phosphor mask types, and their relative brightnesses when displayed at a 1:1 scale on an LCD with an RGB layout.