

Shaders

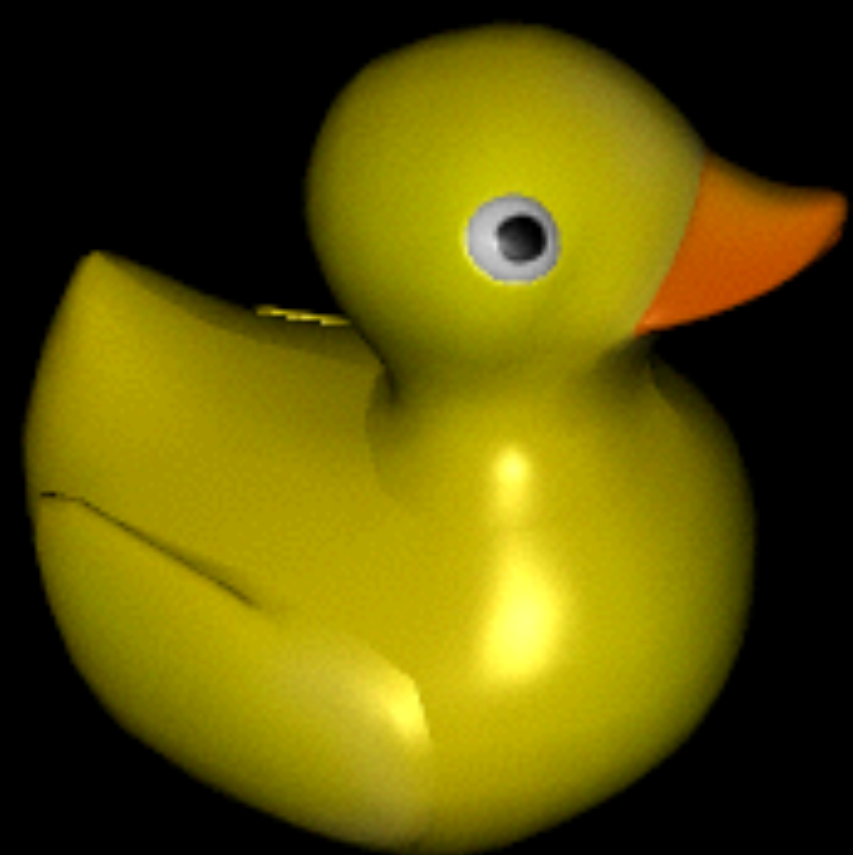
Part 3



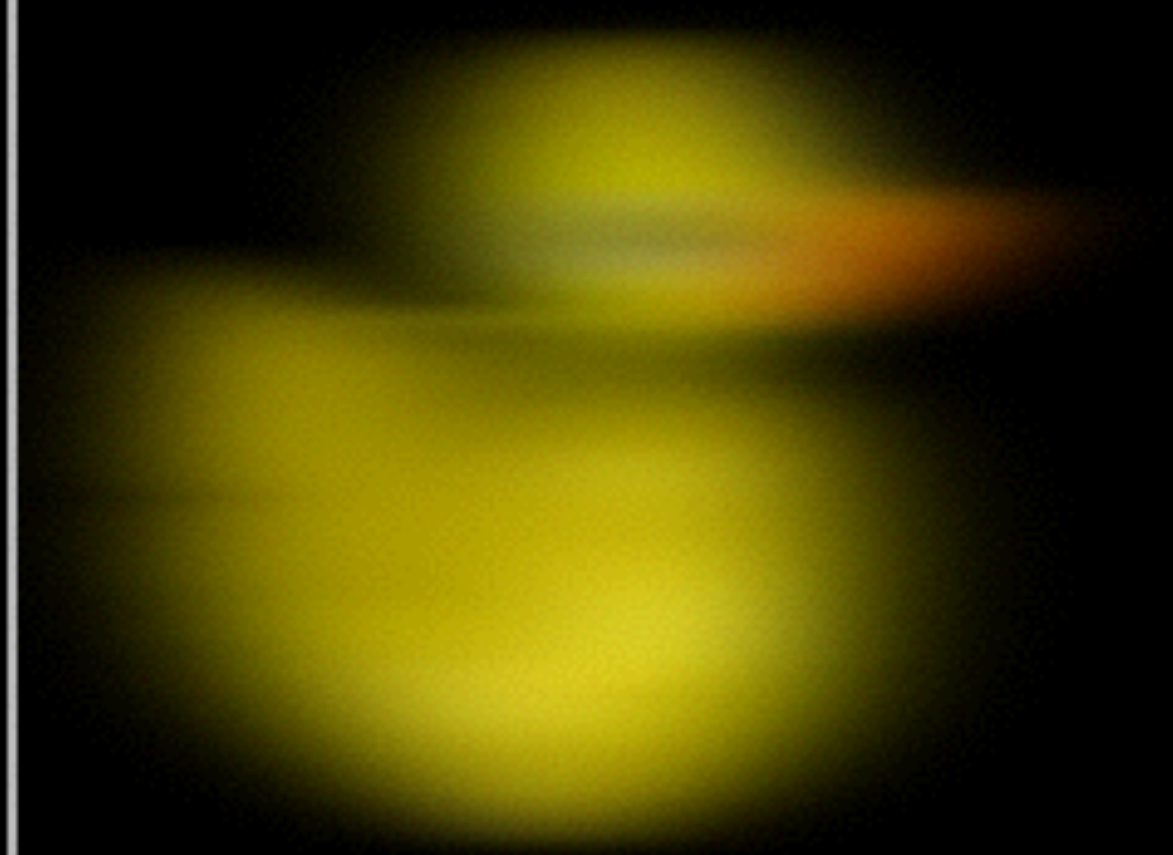
Screen shaders



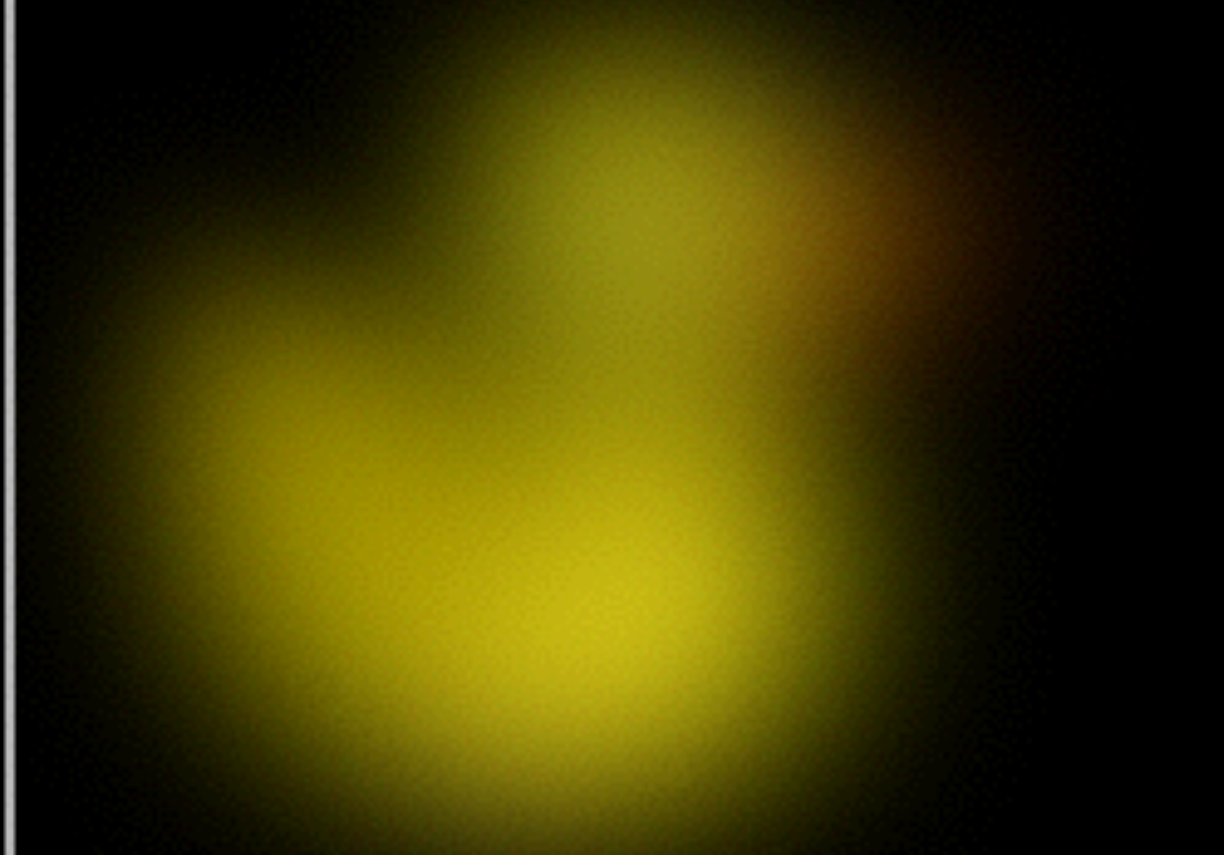




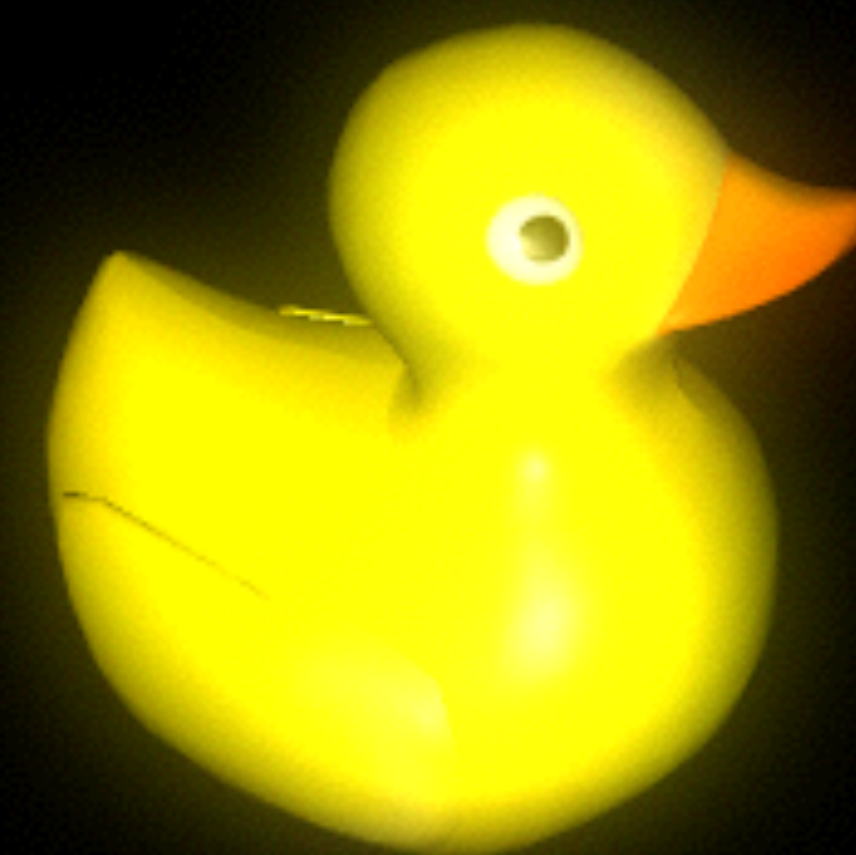
Basic Scene



First Blur Pass (Horizontal)




Second Blur Pass (Vertical)



Final Scene

HOLD  TO EXECUTE BREACH

 OVERLOAD TURRET

 DART OVERLAY
100 %

◇ ALLER AU POINT BRAVO
◇ ÉLIMINER LA SENTINELLE

Attends, j'ai vu quelque chose.

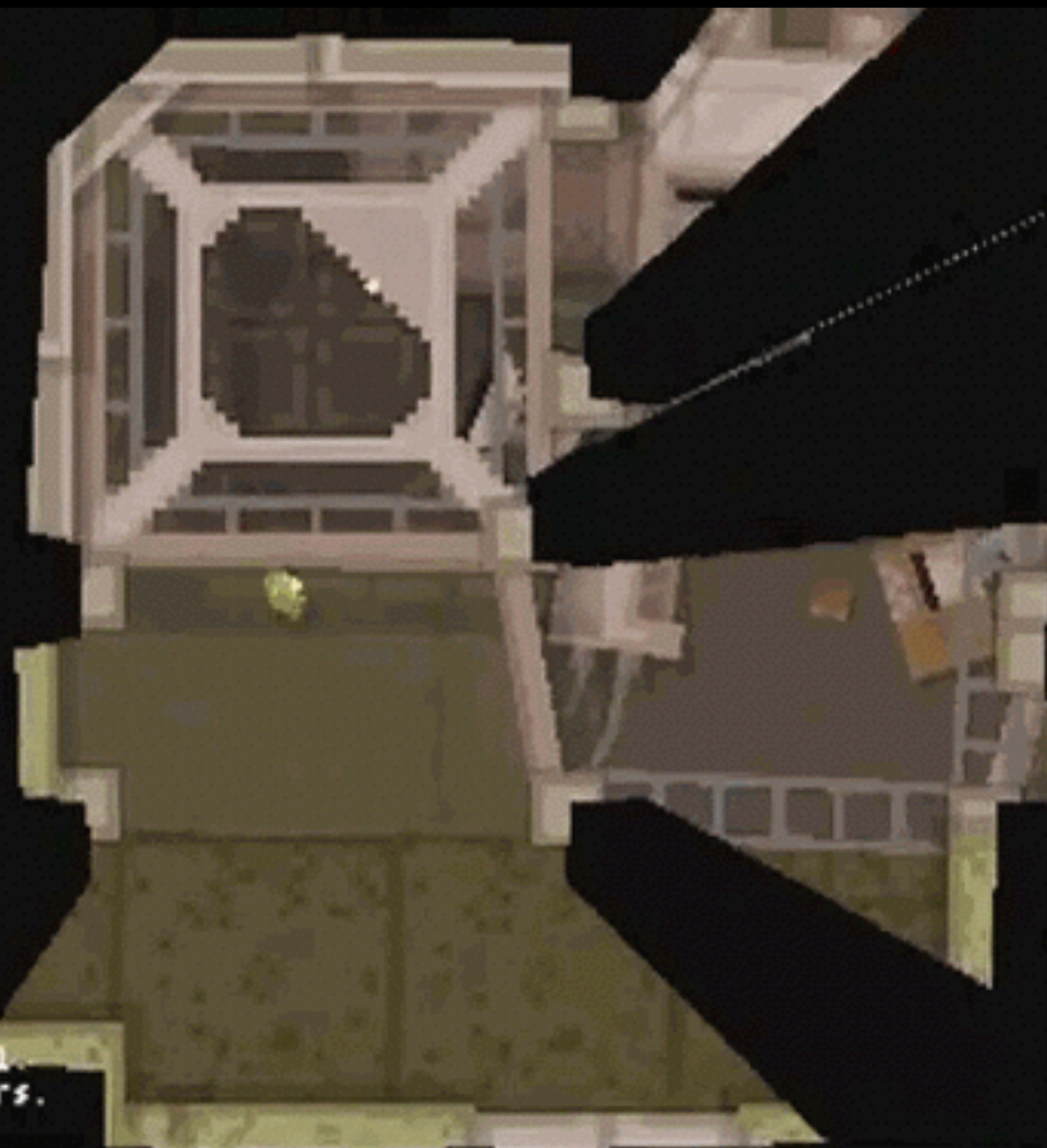
10 / 99
6x0



```
autopstl  
RDX_250 2x  
empty can 4x  
9mmpstl  
agl-1  
tube  
wchip  
nailbox 4x  
RDX_500 2x  
-  
-  
-  
-  
-  
-  
-
```

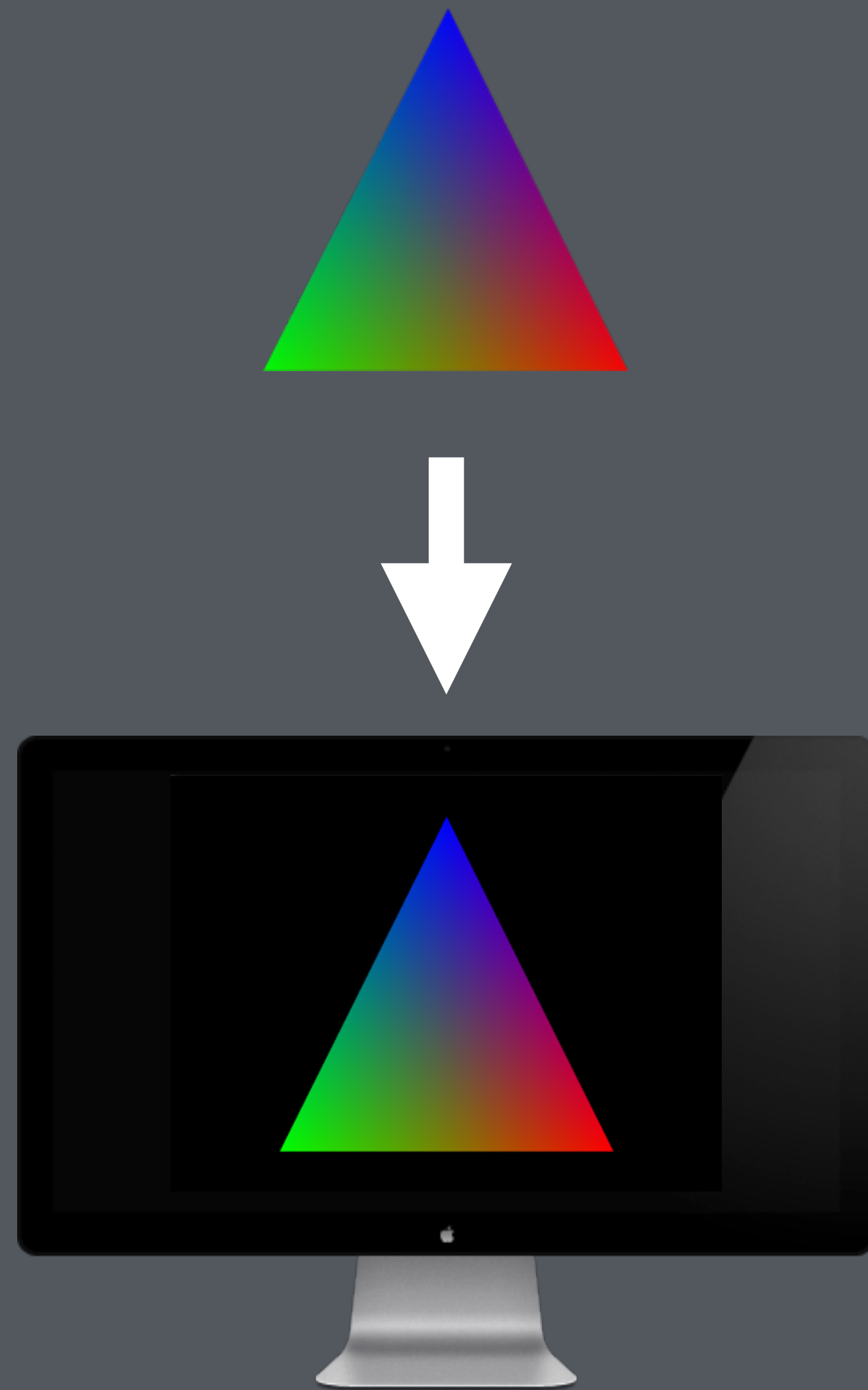
```
ammo =15/5  
health=54/100
```

```
9mm semi-automatic pistol.  
good against small numbers.
```

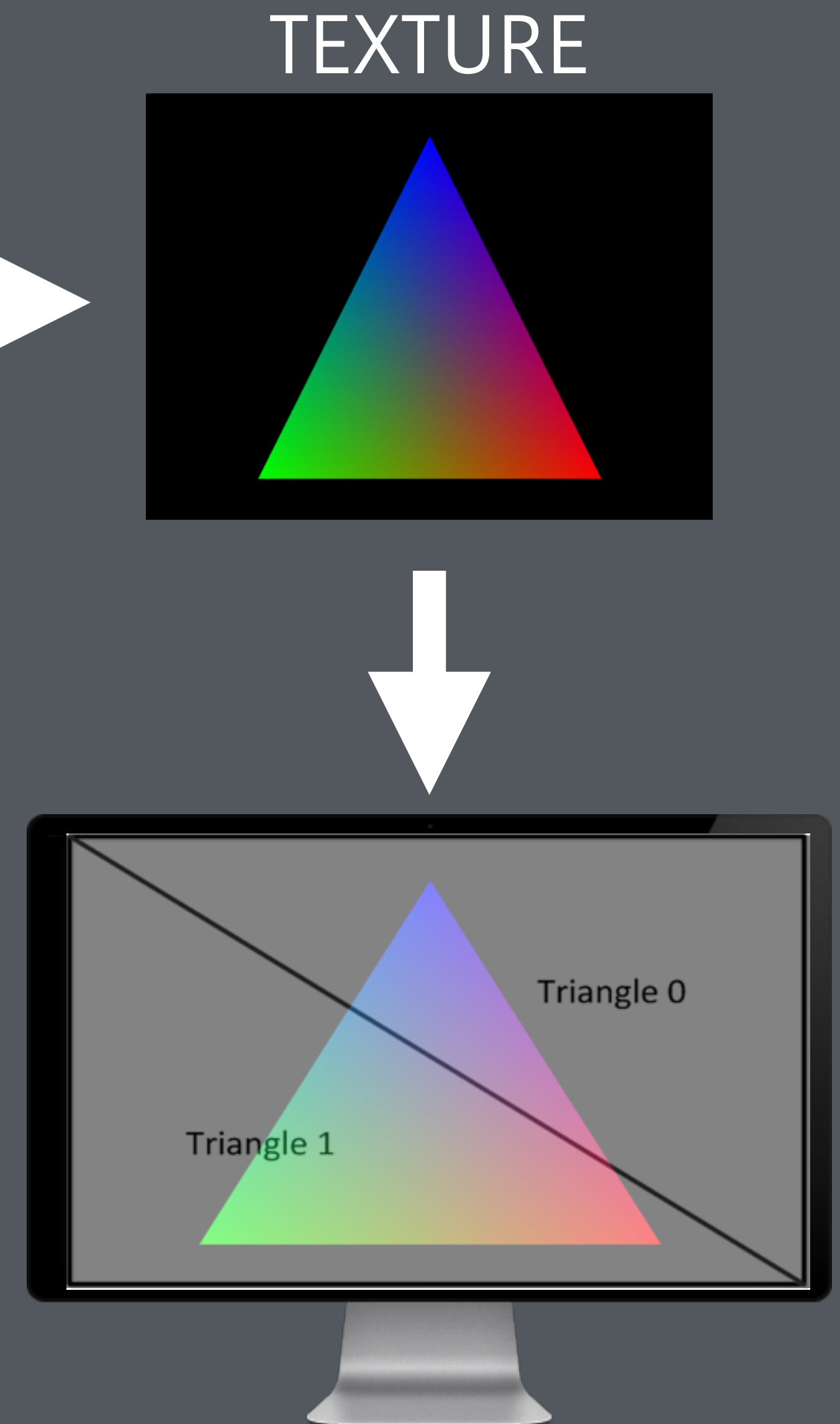


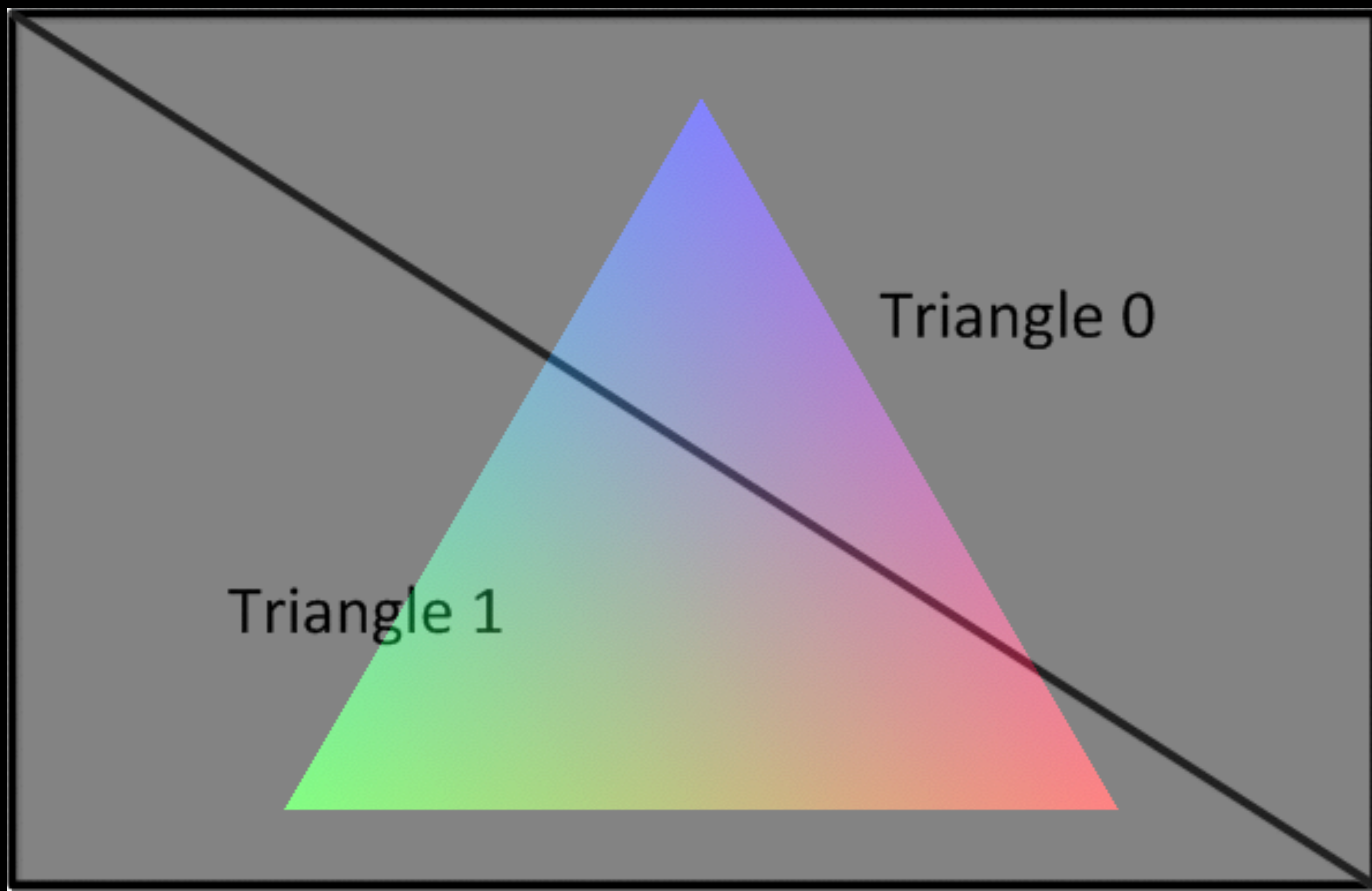
Anatomy of a **screen shader**.

Regular rendering.



Rendering using a screen shader.





Rendering to texture.

Frame Buffer Objects (**FBOs**)

Creating an **FB0**.

Need to define this for SDL before including any header files.

```
#define GL_GLEXT_PROTOTYPES
```


1. **Generate** and **bind** a frame buffer .

```
GLuint framebuffer;  
glGenFramebuffers(1, &framebuffer);  
glBindFramebuffer(GL_FRAMEBUFFER, framebuffer);
```


2. Create an **empty texture** to render to the size of our window.

```
GLuint framebufferTexture;  
glGenTextures(1, &framebufferTexture);  
glBindTexture(GL_TEXTURE_2D, framebufferTexture);  
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, 800, 600, 0, GL_RGB,  
GL_UNSIGNED_BYTE, NULL);  
  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
```


3. Bind the texture to the FBO

```
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0,  
GL_TEXTURE_2D, framebufferTexture, 0);
```


4. If you are using the **z-buffer**, you also need to create a **depth buffer texture**.

```
GLuint depthBufferTexture;  
  
glGenTextures(1, &depthBufferTexture);  
glBindTexture(GL_TEXTURE_2D, depthBufferTexture);  
glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT, 800, 600, 0, GL_DEPTH_COMPONENT,  
GL_UNSIGNED_BYTE, NULL);  
  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);  
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
```

... and bind it to your frame buffer as a depth buffer attachment.

```
glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT24, 800, 600);  
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_TEXTURE_2D,  
depthBufferTexture, 0);
```


Rendering to an FBO.

Bind the FBO **before doing your rendering**. Calling **glClear** after you bind it, will clear the texture.

```
glBindFramebuffer(GL_FRAMEBUFFER, framebuffer);  
glClear(GL_COLOR_BUFFER_BIT);
```

Then, do your rendering as usual.

Rendering the FBO texture to screen.

Vertex shader.

```
attribute vec4 position;  
attribute vec2 texCoord;  
varying vec2 texCoordVar;  
void main()  
{  
    gl_Position = position;  
    texCoordVar = texCoord;  
}
```

No need for projection or modelview matrices as we are just drawing two fullscreen triangles.

Fragment shader.

```
uniform sampler2D diffuse;  
varying vec2 texCoordVar;  
  
void main()  
{  
    gl_FragColor = texture2D( diffuse, texCoordVar);  
}
```

Just draw the texture as is.

After you do your scene rendering, unbind the FBO, use your screen shader program and draw two fullscreen triangles textured with the FBO texture.

You will need to bind the position and tex coordinate attributes of your screen shader

```
glBindFramebuffer(GL_FRAMEBUFFER, 0);
glUseProgram(screenShaderProgram);

glBindTexture(GL_TEXTURE_2D, framebufferTexture);

GLfloat triUVs[] = {
    1.0f, 1.0f,
    1.0f, 0.0f,
    0.0f, 0.0,

    0.0f, 0.0f,
    0.0f, 1.0f,
    1.0f, 1.0f
};

GLfloat tris[] = {
    1.0f, 1.0f,
    1.0f, -1.0f,
    -1.0f, -1.0f,

    -1.0f, -1.0f,
    -1.0f, 1.0f,
    1.0f, 1.0f
};

glVertexAttribPointer(screenPositionAttribute, 2, GL_FLOAT, false, 0, tris);
glEnableVertexAttribArray(screenPositionAttribute);
glVertexAttribPointer(screenTexCoordAttribute, 2, GL_FLOAT, false, 0, triUVs);
glEnableVertexAttribArray(screenTexCoordAttribute);

glDrawArrays(GL_TRIANGLES, 0, 6);

glDisableVertexAttribArray(screenPositionAttribute);
glDisableVertexAttribArray(screenTexCoordAttribute);
```

Some basic screen shaders.

Inverting the screen colors.

```
uniform sampler2D texture;  
varying vec2 texCoordVar;  
  
void main()  
{  
    gl_FragColor = vec4(1.0-texture2D( texture, texCoordVar).xyz, 1.0);  
}
```

Make everything black and white.

```
uniform sampler2D texture;  
varying vec2 texCoordVar;
```

```
void main()  
{  
    vec4 texColor = texture2D( texture, texCoordVar);  
    float brightness = (texColor.x+texColor.y+texColor.z)/3.0;  
    gl_FragColor = vec4(brightness, brightness, brightness, 1.0);  
}
```


Make it wavy!

```
uniform sampler2D texture;  
varying vec2 texCoordVar;  
  
uniform float time;  
  
void main()  
{  
    gl_FragColor = texture2D( texture, vec2(texCoordVar.x+(sin((texCoordVar.y  
+time)*15.0) * 0.1), texCoordVar.y));  
}
```

You'll need to pass in the time elapsed as the “**time**” uniform.

Simple RGB color separation.

```
uniform sampler2D diffuse;
varying vec2 texCoordVar;

void main()
{
    gl_FragColor = vec4(texture2D( diffuse, vec2(texCoordVar.x+0.01, texCoordVar.y)).r,
                        texture2D( diffuse, vec2(texCoordVar.x+0.02, texCoordVar.y)).g,
                        texture2D( diffuse, vec2(texCoordVar.x+0.03, texCoordVar.y)).b,
                        1.0);
}
```


Resolutions and fullscreen.

Fullscreen


```
int SDL_SetWindowFullscreen(SDL_Window *window,  
Uint32 flags);
```

Sets the specified SDL window to fullscreen or windowed mode. Pass flags as:

SDL_WINDOW_FULLSCREEN or **SDL_WINDOW_FULLSCREEN_DESKTOP** for fullscreen mode
or 0 for windowed mode.

```
// set fullscreen  
SDL_SetWindowFullscreen(displayWindow, SDL_WINDOW_FULLSCREEN);
```

```
// set windowed mode  
SDL_SetWindowFullscreen(displayWindow, 0);
```

Enumerating **video modes**.


```
int SDL_GetNumDisplayModes(int displayIndex);
```

Returns the number of display modes for given display index (monitor).

```
// get number of display modes for main monitor  
SDL_GetNumDisplayModes(0);
```

```
int SDL_GetDisplayMode(int displayIndex, int modeIndex, SDL_DisplayMode * mode);
```

Gets the details about a specified mode index and stores it in the SDL_DisplayMode struct pointed to by mode.

Use this in conjunction with **SDL_GetNumDisplayModes** to list available resolutions.

```
for(int i=0; i < SDL_GetNumDisplayModes(0); i++) {  
    SDL_DisplayMode mode;  
    SDL_GetDisplayMode(0, i, &mode);  
    cout << "AVAILABLE RESOLUTION:" << mode.w << "x" << mode.h << endl;  
}
```


Setting a video mode.

In windowed mode.

```
void SDL_SetWindowSize(SDL_Window * window, int w, int h);
```

Sets the window size for a window specified by the SDL_Window pointer.

```
// resize window to 800 x 600  
SDL_SetWindowSize(displayWindow, 800, 600);
```


In fullscreen

```
int SDL_SetWindowDisplayMode(SDL_Window * window, const SDL_DisplayMode * mode);
```

Sets the display mode to the mode specified by the **mode** pointer.

On some platforms, you need to exit and re-enter fullscreen for the new mode to take effect.

```
SDL_DisplayMode mode;  
SDL_GetDisplayMode(0, selectedVideoMode, &mode);  
  
SDL_SetWindowDisplayMode(displayWindow, &mode);  
SDL_SetWindowFullscreen(displayWindow, 0);  
SDL_SetWindowFullscreen(displayWindow, SDL_WINDOW_FULLSCREEN);
```

Don't forget to set your **viewport** and **aspect ratios** to the new resolution!

```
float aspect = (float)currentResolutionX / (float)currentResolutionY;  
projectionMatrix.setPerspectiveProjection(45.0f * PI/180.0f, aspect, 0.1, 200.0);
```

```
glViewport(0,0,currentResolutionX, currentResolutionY);  
projectionMatrix.setOrthoProjection(-aspect, aspect, -1.0, 1.0, -1.0, 1.0);
```


Hiding the cursor.

```
int SDL_ShowCursor(int toggle);
```

Show or hide the mouse pointer. Pass 0 to hide and 1 to show.

```
SDL_ShowCursor(0); // hide the pointer  
SDL_ShowCursor(1); // show the pointer
```